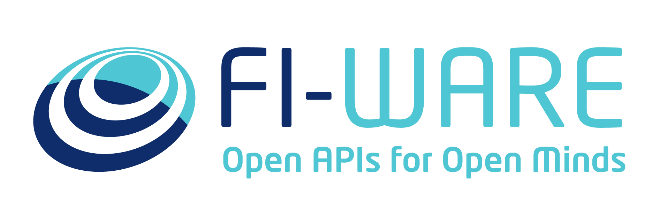
**MNCA IOT SmartCity Technical Architecture**

****

****

**DSI Métropole Nice Côte d’Azur**

**IOT & Smart City Platform**

|  |  |
| --- | --- |
| VERSION: draft | REVISION DATE: 02/11/2017 |

Approval of the Technical Architecture indicates an understanding of the purpose and content described in this deliverable. By signing this deliverable, each individual agrees with the content contained in this deliverable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Approver Name** | **Title** | **Signature** | **Date** |
| Stéphane Roux | ICT Quality Assurance Engineer |  |  |
| Lionel Chaudanson | Responsable technique, IOT & Smart City |  |  |
| Serge Massiera | Directeur des Systèmes d'Information |  |  |
|  |  |  |  |
|  |  |  |  |

[1 Section 1 DOCUMENT SCOPE 6](#_Toc500837464)

[1.1 Context 6](#_Toc500837465)

[1.2 Glossary 8](#_Toc500837466)

[2 Section 2 OVERALL TECHNICAL ARCHITECTURE 9](#_Toc500837467)

[2.1 System Architecture Context Diagram 9](#_Toc500837468)

[2.2 System Architecture Model 13](#_Toc500837469)

[2.2.1 Overall Architectural Considerations 15](#_Toc500837470)

[2.3 System Architecture Component Definitions 18](#_Toc500837475)

[2.3.1 Docker Images Management Components 18](#_Toc500837476)

[2.3.2 Docker Swarm Management Component 19](#_Toc500837477)

[2.3.3 Security Component IDM GE 19](#_Toc500837478)

[2.3.4 Orion Context Broker Component 20](#_Toc500837479)

[2.3.5 MongoDB Cluster Component 20](#_Toc500837480)

[2.3.6 Citus/PostgreSQL Cluster Component 20](#_Toc500837481)

[2.3.7 Galera/MySQL Cluster Component 21](#_Toc500837482)

[2.3.8 Cosmos Big Data Component 21](#_Toc500837483)

[2.3.9 Knowage Analytics Component 22](#_Toc500837484)

[2.3.10 CKAN Open Data Component 22](#_Toc500837485)

[2.3.11 IDAS IOT Components 23](#_Toc500837486)

[2.3.12 IOT Broker Component 24](#_Toc500837487)

[2.3.13 IOT Discovery Component 24](#_Toc500837488)

[2.3.14 CEP Component 24](#_Toc500837489)

[2.3.15 Application Mashup Components 24](#_Toc500837490)

[2.3.16 Business Ecosystem Component 25](#_Toc500837491)

[2.3.17 Development Components 26](#_Toc500837492)

[2.3.18 Monitoring Components 26](#_Toc500837493)

[2.3.19 Poi Data Provider Component 27](#_Toc500837494)

[2.3.20 GIS Data Provider Component 27](#_Toc500837495)

[2.3.21 Load Balancing Component 28](#_Toc500837496)

[3 Section 3 SYSTEM ARCHITECTURE DESIGN 28](#_Toc500837497)

[3.1 Docker Images Management Component 28](#_Toc500837498)

[3.1.1 Component Functions 28](#_Toc500837499)

[3.1.2 Technical Considerations 29](#_Toc500837500)

[3.1.3 Selected Product(s) 29](#_Toc500837501)

[3.1.4 Selection Rationale 29](#_Toc500837502)

[3.1.5 Architecture Risks 29](#_Toc500837503)

[3.2 Docker Swarm Manager Component 29](#_Toc500837504)

[3.2.1 Component Functions 29](#_Toc500837505)

[3.2.2 Technical Considerations 30](#_Toc500837506)

[3.2.3 Selected Product(s) 30](#_Toc500837507)

[3.2.4 Selection Rationale 31](#_Toc500837508)

[3.2.5 Architecture Risks 31](#_Toc500837509)

[3.3 Security Component IDM GE 31](#_Toc500837510)

[3.3.1 Component Functions 31](#_Toc500837511)

[3.3.2 Technical Considerations 33](#_Toc500837512)

[3.3.3 Selected Product(s) 33](#_Toc500837513)

[3.3.4 Selection Rationale 33](#_Toc500837514)

[3.3.5 Architecture Risks 33](#_Toc500837515)

[3.4 Orion Context Broker Component 34](#_Toc500837516)

[3.4.1 Component Functions 34](#_Toc500837517)

[3.4.2 Technical Considerations 34](#_Toc500837518)

[3.4.3 Selected Product(s) 35](#_Toc500837519)

[3.4.4 Selection Rationale 35](#_Toc500837520)

[3.4.5 Architecture Risks 36](#_Toc500837521)

[3.5 MongoDB Component 36](#_Toc500837522)

[3.5.1 Component Functions 36](#_Toc500837523)

[3.5.2 Technical Considerations 36](#_Toc500837524)

[3.5.3 Selected Product(s) 37](#_Toc500837525)

[3.5.4 Selection Rationale 37](#_Toc500837526)

[3.5.5 Architecture Risks 37](#_Toc500837527)

[3.6 Citus/PostgreSQL cluster Component 37](#_Toc500837528)

[3.6.1 Component Functions 37](#_Toc500837529)

[3.6.2 Technical Considerations 37](#_Toc500837530)

[3.6.3 Selected Product(s) 38](#_Toc500837531)

[3.6.4 Selection Rationale 38](#_Toc500837532)

[3.6.5 Architecture Risks 38](#_Toc500837533)

[3.7 Galera/MySQL cluster Component 39](#_Toc500837534)

[3.7.1 Component Functions 39](#_Toc500837535)

[3.7.2 Technical Considerations 39](#_Toc500837536)

[3.7.3 Selected Product(s) 39](#_Toc500837537)

[3.7.4 Selection Rationale 39](#_Toc500837538)

[3.7.5 Architecture Risks 39](#_Toc500837539)

[3.8 Cosmos Big Data Components, Cygnus & Comet 39](#_Toc500837540)

[3.8.1 Component Functions 39](#_Toc500837541)

[3.8.2 Technical Considerations 42](#_Toc500837542)

[3.8.3 Selected Product(s) 43](#_Toc500837543)

[3.8.4 Selection Rationale 43](#_Toc500837544)

[3.8.5 Architecture Risks 44](#_Toc500837545)

[3.9 Knowage Analytics Component 44](#_Toc500837546)

[3.9.1 Component Functions 44](#_Toc500837547)

[3.9.2 Technical Considerations 45](#_Toc500837548)

[3.9.3 Selected Product(s) 45](#_Toc500837549)

[3.9.4 Selection Rationale 45](#_Toc500837550)

[3.9.5 Architecture Risks 45](#_Toc500837551)

[3.10 CKAN Open Data Component 45](#_Toc500837552)

[3.10.1 Component Functions 45](#_Toc500837553)

[3.10.2 Technical Considerations 45](#_Toc500837554)

[3.10.3 Selected Product(s) 45](#_Toc500837555)

[3.10.4 Selection Rationale 45](#_Toc500837556)

[3.10.5 Architecture Risks 45](#_Toc500837557)

[3.11 IDAS IOT Components 45](#_Toc500837558)

[3.11.1 Components Functions 45](#_Toc500837559)

[3.11.2 Technical Considerations 48](#_Toc500837560)

[3.11.3 Selected Product(s) 48](#_Toc500837561)

[3.11.4 Selection Rationale 48](#_Toc500837562)

[3.11.5 Architecture Risks 49](#_Toc500837563)

[3.12 CEP Component 49](#_Toc500837564)

[3.12.1 Component Functions 49](#_Toc500837565)

[3.12.2 Technical Considerations 49](#_Toc500837566)

[3.12.3 Selected Product(s) 49](#_Toc500837567)

[3.12.4 Selection Rationale 49](#_Toc500837568)

[3.12.5 Architecture Risks 49](#_Toc500837569)

[3.13 Application Mashup Components 49](#_Toc500837570)

[3.13.1 Component Functions 50](#_Toc500837571)

[3.13.2 Technical Considerations 50](#_Toc500837572)

[3.13.3 Selected Product(s) 50](#_Toc500837573)

[3.13.4 Selection Rationale 51](#_Toc500837574)

[3.13.5 Architecture Risks 51](#_Toc500837575)

[3.14 Business Ecosystem GE Components 52](#_Toc500837576)

[3.14.1 Components Functions 52](#_Toc500837577)

[3.14.2 Technical Considerations 53](#_Toc500837578)

[3.14.3 Selected Product(s) 53](#_Toc500837579)

[3.14.4 Selection Rationale 53](#_Toc500837580)

[3.14.5 Architecture Risks 53](#_Toc500837581)

[3.15 Development Components 53](#_Toc500837582)

[3.15.1 Component Functions 53](#_Toc500837583)

[3.15.2 Technical Considerations 53](#_Toc500837584)

[3.15.3 Selected Product(s) 54](#_Toc500837585)

[3.15.4 Selection Rationale 54](#_Toc500837586)

[3.15.5 Architecture Risks 54](#_Toc500837587)

[3.16 Monitoring Components 54](#_Toc500837588)

[3.16.1 Component Functions 54](#_Toc500837589)

[3.16.2 Technical Considerations 54](#_Toc500837590)

[3.16.3 Selected Product(s) 54](#_Toc500837591)

[3.16.4 Selection Rationale 55](#_Toc500837592)

[3.16.5 Architecture Risks 55](#_Toc500837593)

[4 Section 4 System Construction Environment 55](#_Toc500837594)

[4.1 Preliminary test Environment 55](#_Toc500837595)

[4.2 Global system vision 55](#_Toc500837596)

[4.3 Development Environment 56](#_Toc500837597)

[4.3.1 Developer Workstation Configuration 56](#_Toc500837598)

[4.3.2 Supporting Development Infrastructure Configuration 56](#_Toc500837599)

[4.4 QA Environment 56](#_Toc500837600)

[4.4.1 QA environment Configuration 56](#_Toc500837601)

[4.4.2 Supporting QA Infrastructure Configuration 56](#_Toc500837602)

[4.5 Production Environment 57](#_Toc500837603)

[4.5.1 Production environment Configuration 57](#_Toc500837604)

[4.5.2 Supporting Production Infrastructure Configuration 57](#_Toc500837605)

# Section 1 DOCUMENT SCOPE

## Context

***Document Scope*** *describes the context and the goals of this document in a narrative.*

This document describes the Technical Architecture of the IOT Smart City Platform that:

* satisfies business requirements as documented in the roadmap (ref to provide),
* satisfies technical, operational and transitional requirements.

New technologies, particularly digital technologies, allow the development of new services and new uses for improving the quality of life in the respect the environment in today’s metropolis.

With the advent of IOT, Big data and AI, the concept of smart city can come to reality.

To anticipate these paradigm shifts, the metropolis of Nice launched in 2008 under the leadership of Mr Estrosi, an initiative multichannel scale that aims to achieve the digital transformation of the territory in the PACA region.

The main idea behind this major initiative lies in the establishment of complex synergies between local authorities, the university (Research and education) and economic actors (industry and start-ups).

These synergies are intended to develop the best technologies and uses them more successful in the service to citizens.

Indeed, advanced technologies and the use intelligent interest data should allow today, to improve the quality of living, well-being and the functioning of the urban unit in the constraints of sustainable development.

The challenge for the Nice metropolis is also to maintain control of these while promoting economic spin-offs and job creation.

The recent obtaining of the label IDEX (January 2016) ideally positions the metropolis of Nice and the Côte d'Azur University (UCA) in the national pole of excellence and therefore European.

The IDEX and the Nice metropolis have chosen to strengthen, in particular three complementary axes: smart-city, health / silver economy and telco / electronics.

These cutting-edge technological and societal initiatives are part of an organization

rationality of the territory:

* The plain of Var for Eco-Valley / smart-city
* Pasteur area for health / silver economy
* Sophia-Antipolis for electronics / Telco / Pharmaceutical /

The coherence and functionality of the whole are strategically driven by the public transport network including the T2 and T3 tram lines which will connect directly to the airport and train station, eco-valley and Arenas stadium.

Local and university authorities have therefore established a single framework facilitating the functional translation of the most advanced concepts from laboratories to the end user. Small and large businesses have all the keys to development of innovations and their economic projects.

In order, to make this transformation into a smart city, a reality, Nice Metropolis after a period of POC, have decided to reach a new level.

The main objectives for this transformation are among:

* Provide a coherent and efficient framework for application development
* App store and Market Place
* Open data
* Labeling of applications
* Better visibility for users
* Aggregate the different sensor subnetworks into a metropolitan collection backbone network
* Adopt a logic of optimization and rationalization
* Capitalize on existing networks managed by the ISD (470 km of fiber optics, radio bridges, VPN-M2M)
* Standardize and store information to facilitate collaboration between operational departments
* Share and enrich these data by experimenting with industrialists, laboratories, researchers, communities, citizens, ...
* Update continuously various predictions by integrating real-time data
* Develop an hyper-vision system for the management of exceptional events (floods, demonstrations, disorders, ...)

Therefore, after a study conducted by Eridanis on the state of art of IOT platforms for Smart City, a roadmap has been provided for the setup of such a system, based mainly on Fiware.

In the referenced roadmap document, the needs have been described through the following global schema:

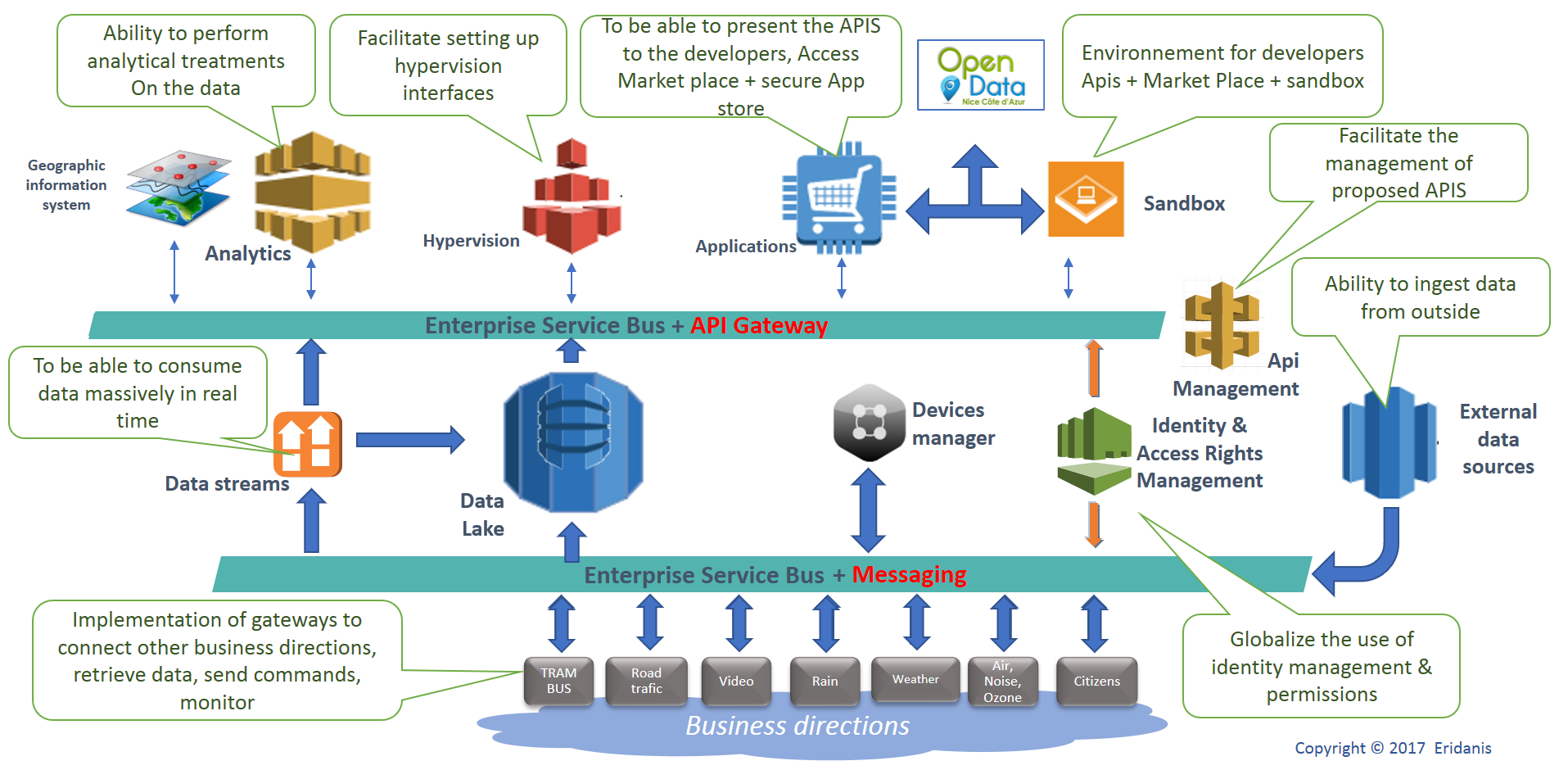


Figure 1: Target Logical Smart City Platform

The goal of this Technical Architecture is to define the technologies, products, and techniques necessary to install and support the system, and to ensure that the system components are compatible and comply with the enterprise-wide standards and direction defined by Nice Côte d’Azur metropolis (hereafter referred as MNCA).

This document will also:

* Identify and explain the risks inherent in this Technical Architecture;
* Define baseline sizing, archiving and performance requirements;
* Identify the hardware and software specifications for the Development, Testing, QA and Production environments;
* Identify monitoring and support tools for maintenance in operational condition
* Define procedures for data migration among the environments (current data warehouse).

## Glossary

**SOA:** Service Oriented Architecture, is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.

**Micro Services Architecture**: Microservices is a variant of the Service-Oriented Architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services should be fine-grained and the protocols should be lightweight, it naturally enforces a modular structure.

**FIWARE**: is an independent open community whose members are committed to materialize the FIWARE mission, that is: “to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors”.

**Fiware platform**: is a platform providing a simple but powerful set of APIs that ease the development of Smart Applications in multiple vertical sectors: <https://www.fiware.org/>, it is based mainly on a Micro Services Architecture, where components expose RESTful APIs.

**IOT**: Internet Of Things

**OMA**: [Open Mobile Alliance](http://www.openmobilealliance.org/) : OMA is a non-profit organization that delivers open specifications for creating interoperable services that work across all geographical boundaries, on any bearer network.

**TM Forum**: [TM Forum](https://www.tmforum.org/) is the global industry association that drives digital transformation of the communications industry through collaboration.

**GE**: Generic Enabler, Fiware used terminology to design an essential module of the platform

**API**: Application Programming Interface

**RESTful API**: also referred to as a RESTful web service, it is based on representational state transfer ([REST](https://en.wikipedia.org/wiki/Representational_state_transfer)) technology, an architectural style and approach to communications often used in web services development as an alternative to [SOAP](https://en.wikipedia.org/wiki/SOAP).

**Context Broker**: According to Gartner:

“A context broker is a service that is designed to gather reachable context data of a variety of types, sources and velocity. It then applies conditioning, integration, rules and analytics to derive the reduced prepared context data, actionable at a point of business decision by a system or a human.”

Functionally speaking, context enables the data to be empowered by the interaction with other pieces of data. In a nutshell it can be consider as a data exchange bus, providing access to context information through publish/subscribe paradigm (this is kind of “MQTT” for context data, through RESTful API).

**NGSI**: New Generation Services Interface, [OMA referenced API specifications](http://www.openmobilealliance.org/release/NGSI/) defined ([NGSI V9 and V10](https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model)) and used with amendments by Fiware, for use through Context Broker, as the dorsal for the platform: NGSI V1 and now [NGSI V2](http://fiware.github.io/context.Orion/api/v2/stable/) which now the reference api version.

**Docker**: is a virtualization software technology providing containers to run software applications in isolation: [Docker site](https://www.docker.com/), [Docker Documentation](https://docs.docker.com/)

**Swarm**: [Docker Swarm](https://docs.docker.com/engine/swarm/) provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single virtual Docker engine.

**VRF**: Virtual Routing and Forwarding network component.

**GDPR**: General Data Protection Regulation

**RIA**: Rich Internet Application

**UI**: User Interface

**Ontology**: formal specification of a conceptualization, used to explicitly capture the semantics of a certain reality

**SPOF**: Single Point Of Failure

# Section 2 OVERALL TECHNICAL ARCHITECTURE

## 2.1 System Architecture Context Diagram

The **Smart city plarform Architecture Context Diagram** provides the “big picture” view of the global system’s architecture, and puts it in context with the rest of the Performing Organization’s systems portfolio, illustrating how the system’s hardware and software platforms fit into the existing environment.

The MNCA IOT Smart City Platform will be deployed upon the 3 data centers owned and managed by the MNCA DSI

* The main data center is Phoenix,
* The secondary is Bosio (currently underworks for revamping for 6 months),
* The third one (the smallest) is Biscarra, (the configuration has to be reconsidered but after Bosio .
* The hardware infrastructure is based on bare metal servers (Cisco and a few remaining HP)
* The servers are blades in enclosures, 3 enclosures per cabinet
* The machines are managed with vSphere and VMWare ESX as virtualization layer, vCenter, vMotion.
* There is a redundant SAN (EMC), dispatched in main and secondary DC. (capacity ?)
* It exists also a redundant NAS system for low I/O file storage.

MNCA Network is managed with a cluster of firewall/routers, managing the separation in 4 different virtual networks

* Public DMZ for applications
* Private DMZ
* LAN
* Field VLANs for IOT gateways, devices and sensors

There’s several Load balancing F5 Big IP appliances at strategic points to distribute load to data centres.



Figure 2: MNCA Infrastructure Global View

The previous schema shows globally the IT and network infrastructure in place for MNCA.

Since servers are dispatched through the 4 VLANs independently from the cabinets, the schema is not the true image of cabinets repartition in the different VLANs

The existing IT services for MNCA are deployed through this available infrastructure.

The Smart City platform will as well be deployed on the same data center infrastructure, but a new DMZ network will be created to host most of the Smart City platform components.

IOT components such as IOT agents, will be deployed under the IOT field dedicated VLANs (industrial VRF).

Nevertheless, some new hardware infrastructure will most probably be required, to provide network, computing power and storage.

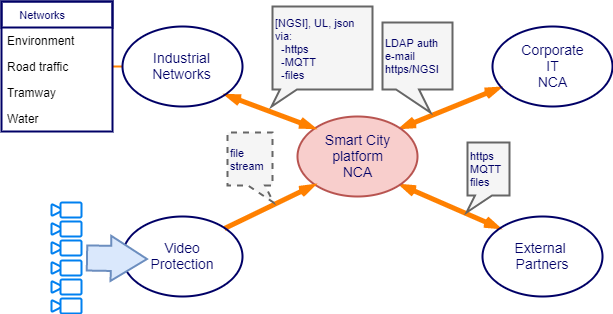


Figure 3: Smart City Platform Interaction with other systems

The envisaged interaction between those services and the smart City platform is for the moment limited to:

* use MNCA mail services (outgoing/incoming)
* connect existing data sources to the new system instead of the current used data warehouse
* use global MNCA security system based on Microsoft Adam (LDAP server), when possible
* the platform will interface to several industrial networks to gather information from field such as: road traffic, water management, sanitation network, environmental monitoring (noise, air quality, ozone), public transport (buses, tram), weather, video monitoring, citizens, buildings monitoring...

After a selection process it has been agreed that the MNCA smart city platform will be mainly based on [FIWARE](https://www.fiware.org/)

The Fiware community propose a set of modules called Generic Enablers, which provide the essential software bricks to set a complete smart city platform, like a kind of “Lego”.

The FIWARE architecture is by nature open, and based on open source modules available in a [dedicated catalog](https://catalogue.fiware.org/)

The most important to mention is that FIWARE has deliberately chosen to go towards a container based architecture, this limits the dependency with Openstack initially thought as the preferred cloud stack for deploying Fiware platforms using virtual machines.

Fiware started in 2014, which was just the beginning of the “Dockerization” of the server software world, the Fiware consortium couldn’t stay without going towards container based deployment, Docker is now an officially supported way to deploy Fiware components in a very clear and easy way.

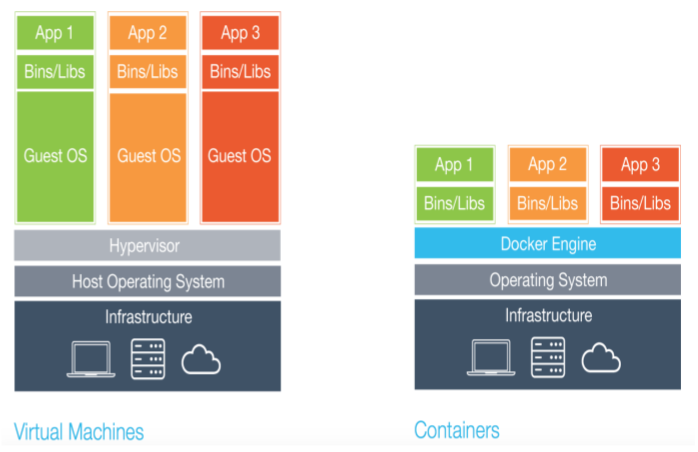


Figure 4: Docker compared to Virtual Machines

MNCA have the opportunity to introduce new way to use its infrastructure using container based architecture, by choosing to use Docker to deploy its own Fiware plarform as a reference Smart City Platform.

The global architecture of the MNCA Fiware platform will rely on a Docker Swarm composed of nodes dispatched in the different VLANs.

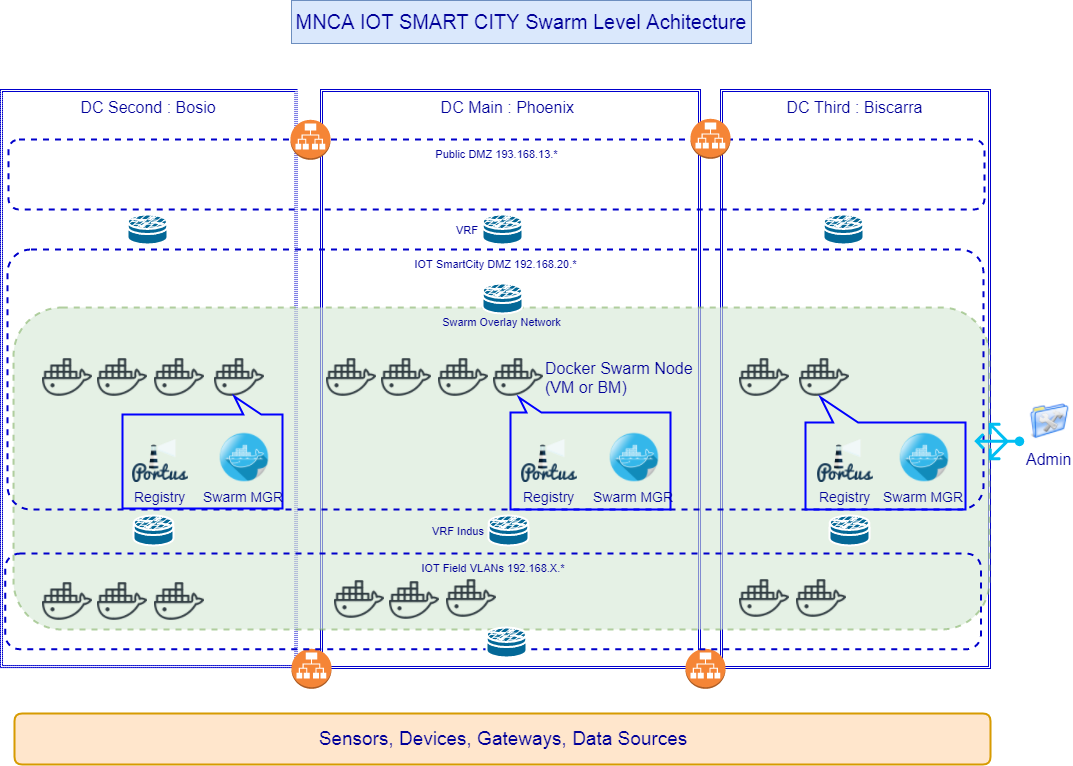


Figure 5: Docker swarm deployment for MNCA Smart city Fiware platform

A Docker swarm (Cluster management integrated with Docker Engine) solution has been chosen in a first step because of the ease of use and configuration.

The learning curve of Docker container orchestration tool: Swarm, is less sloping than other solution such as Mesos, or Kubernetes (the open source declination of the Google internal orchestration tool: Borg), which sit at the top in term of features, openness, and adoption, among cloud orchestration tools today’s landscape.

Even it is the right choice to start with simple Docker Swarm, it will be easy to add later another orchestration tool like Kubernetes, since it will be proposed as the Swarm complement in Docker Enterprise distribution ([very recent news](https://thenewstack.io/docker-fully-embraces-kubernetes/)), the upgrade feasibility is guaranteed.

Then the MNCA FIWARE smart city platform will be deployed in a global docker swarm covering the 3 data centers.

The repartition of the traffic between the data centers will be managed by F5 BigIP load balancing appliance.

However, to minimize configuration task on load balancer (limit to load repartition between data centers of main categories of services), for the lower granularity of applications, and internal components of the platform, simple load balancing and fail-over mechanisms are managed automatically by the Docker Swarm (number of instances of a service).

For the deployment of the platform over a Swarm, we still need to provision virtual machines or bare metal servers and add it to the swarm.

For the moment VM are created using VMWare vSphere, it’s simple, but it comes with a cost per each created VM…

VMware proposes vSphere® Integrated Containers, which allows to manage containers from vSphere, but it doesn’t change the cost for the deployed VMs a license fee is still applied.

A solution is proposed: [VMWARE INTEGRATED OPENSTACK](https://www.vmware.com/fr/products/openstack.html), (VIO), which allows to leverage OpenStack from VMWare vSphere, the infrastructure is managed using OpenStack tools, but there’s still license fee to pay per CPU (VIO 4.0 per-CPU license and Support and Subscriptions (SnS)).

Despite this license fees issue, VIO offers Container execution on OpenStack: for customers seeking to modernize their traditional applications or to design new ones, based on containers, VMware Integrated OpenStack offers **Kubernetes** support ready for production with pooling and persistent storage (volumes). This solution ready for use for who want to run containers on OpenStack and exploit VMware's virtualization technologies.

The choice to use the current way or VIO with vSphere, to manage the platform infrastructure need to be assessed with induced costs.

Anyway, using only pure OpenStack tools to manage the platform infrastructure is a solution to consider at least to lower TCO in terms of license fees.

## System Architecture Model

The **System Architecture Model** represents the various architecture components that comprise the system, and shows the important interrelationships.

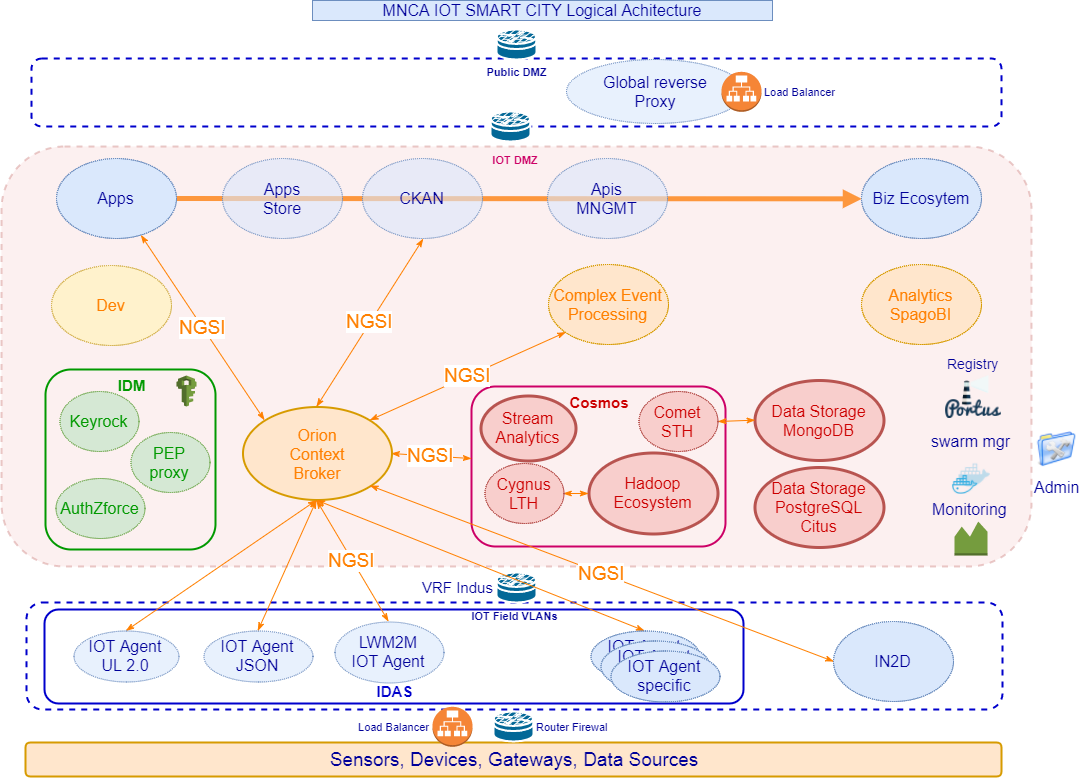


Figure 6: MNCA IOT Smart City Platform logical architecture

* Not all the relations between components have been represented in the above schema to preserve readability.
* Most of the components provide interfaces through RESTful API over HTTP/HTTPS, so that the platform is SOA (Service Oriented Architecture), and micro services based.
* The central component that provides the main “context data exchange bus” is the Orion Context Broker, this is the Fiware Generic Enabler (GE) to consider as the cornerstone of the platform.
* As any interaction with this component is done using NGSI RESTful APIs, this is the main communication stream in the platform, that is represented on the above figure.
* As well the transversal arrow to Biz ecosystem GE, represents the link of each component to it for platform assets catalog management and monetization.
* Are not represented the relations to the identity manager (IDM GE) component, which are generally used to authorize access to the component through a proxy to identity manager.
* The aim with the IDM (Identity Management) component, is to try to offer a centralized SSO management over the whole platform, allowing the central control of access authorization to platform components.
* The data storage component is not a Fiware component, this should be the SQL data tank of any Fiware module using SQL database, in case it’s impossible to change (a lot of Fiware modules rely on MySQL database) to PostgreSQL, we can use MariaDB.
* The general idea with the data storage components in the representation is to emphasis the fact that all the databases used by Fiware modules should be centralized in some clusters based on needed database server, which is true for SQL database and NoSQL database, so that relation links have not been represented on the above schema
* Inside the DEV component on the above figure are the Paas and the tools needed to develop applications that will use the Fiware ecosystem, this encompass development tools but also QA tools for testing and validation purpose, which can have specific needs in term of access to data. Most probably for development and QA, it will be required to deploy some Fiware components or even a complete platform for staging.
* A global reverse proxy and load balancer (F5 BigIP) will be the main entry point to the smart city platform.

### 2.2.1 Overall Architectural Considerations

*The* ***Overall Architectural Considerations*** *section defines how additional technical requirements have been addressed by the architecture. items in this section include:*

* *Deployment strategy*
* *Security Strategy*
* *Availability requirements*
* *Performance requirements*
* *Accessibility*
* *Database management*
* *Storage policy*
* *Transaction volumes*
* *Concurrent users & devices*
* *Data import and export*
* *Disaster recovery*
* *GDPR*

#### Deployment Strategy

As stated previously, the MNCA Smart City platform will be deployed using Docker containers in a global swarm (group of Docker engines).

In this first step Docker engines will be deployed on Virtual machines dispatched on the 3 data centers (when they will be available, for the first step installation the platform will be deployed on the main data center).

The different modules will be grouped in services and stacks (Docker concepts).

Database servers, will be deployed in clusters, at disposal for use by other components of the platform.



#### Security Strategy

Application security is achieved through a global identity and access rights management component the Fiware Identity Management GE, providing SSO and secure access to other components of the platform, according to requester authorizations.

The identity management component will be connected to the MNCA LDAP (MS ADAM based).

By using [Docker Swarm's secrets](https://blog.docker.com/2017/02/docker-secrets-management/), component’s secrets (passwords, tokens, SSH private keys, certificates) can be safely stored and used in services composition files without compromising security.

Docker Images used for the platform will be locally managed in a secured registry.

#### Availability requirements

All the components used in the platform will be deployed in a way providing high availability.

The services and the associated databases should ideally be deployed on the 3 data centers,

The third data center remaining off-line most of time, should host replicas of the databases, updated in real-time.

The use of Docker provides an easy way to deploy highly available services, by deploying several containers on several nodes of a swarm, using docker compose files or docker stack files.

#### Performance requirements

Any of the components will be scalable by adding some instance of containers to scale-up the service.

Through the use of Docker it is very easy to achieve by changing services configuration on the fly.

#### Accessibility

Nothing specific is included in Fiware for accessibility, this item is to be considered at application UI level, most probably using accessibility features from operating systems or internet browsers that host the client application.

#### Database management

Different kind of databases are used among the Fiware components:

PostgreSQL, MySQL, MongoDB, and Hadoop cluster.

To simplify the maintenance:

PostgreSQL will replace MySQL in each component where it’s possible, unless it’s not possible to change easily to PostgreSQL, [MariaDB](https://mariadb.org/) should replace MySQL without any change.

To rationalize the architecture, each kind of database service will be deployed once in the platform, shared by the client components.

Each of the database service will be based on a cluster:

* This is native architecture for MongoDB
* PostgreSQL databases will be based on a Citus database cluster
* MySQL/MariaDB databases will be based on a Galera database cluster

May be thinking to master data management at the occasion of this project is an opportunity to consider

In particular, to reach GDPR conformity more easily, by using appropriate tools to manage data and the access rights.

#### Transaction volumes

Write transactions volume:

In the excel document about data input traffic, the number of transactions per day for data acquisition has been evaluated around 620 000, this is a very low traffic.

A data acquisition transaction consists in the transformation of the message to NGSI, the update of the data in real time storage and insert it in history data store.

the top peak known demand is for tramway data (216 data/28 trains/every minute), for existing T1 line,

even we project to 100 trains with the 3 lines working, it’s less than 2 transactions per seconds.

Read transaction volume:

To give a global evaluation at this step of the project is not easy, since it depends mostly on the type of application, and the traffic they should generate

The database clusters will be easily horizontally scalable, to support read transaction increase in case of need.

#### Storage Sizing

Depending on the component, some storage volumes have to be used, to persist data outside of the container itself.

For most of components except database servers, the storage needs are limited to the container image footprint (from 40MB for simple to 1GB for very complex images), and some configuration files (mandatory for the container to run, but negligible in size).

In section 3, for each component indication about the storage requirements will be assessed.

For database clusters, the problematic is different and is described in section 3 as well.

#### Storage Persistence

As the platform uses containers, managed as services with Docker, the containers are transient, and if a new container is started, any file inside will be lost, so it’s necessary to use Docker volume storage to persist the data to keep.

If the volume is created on

The first simple idea should be to use Docker volumes mapped to NFS shared volumes.

Another solution should be to use solutions like [Flocker](https://flocker.readthedocs.io/en/latest/supported/index.html) or [Rex-Ray](http://rexray.readthedocs.io/en/stable/user-guide/storage-providers/), that are able to work with different network storage solutions, among them EMC, in use in MNCA, those solutions are able to replicate the data in the new container in a transparent way.

#### Concurrent user & devices

The scalability of the platform using containers, allows to virtually support any numbers of concurrent users and devices, by deploying more instances of services, providing the provisioning of enough computing nodes as virtual machines or bare metal servers to absorb the load.

By starting with a simple configuration on 3 or 4 nodes, we will evaluate performance through some simple use cases scenarios, with a monitoring system beside which stores history.

#### Data import, export and models

Data existing in the current data warehouse will be imported into the new platform,

The main concern remains the current data model adaptation to existing [Fiware data models](https://www.fiware.org/data-models/) and eventually propose amendments when needed.

This is currently under study, in order to be able to propose a table of correspondence between the existing data source attributes and the attributes defined in the schema for the destination data object (an NGSI entity), that, if possible, match an existing Fiware data model.

In case new attributes need to be added to a Fiware proposed model, or new data model, the proposal should be sent to Fiware for validation, respecting the given guidelines on [How To Contribute](https://github.com/fiware/dataModels#how-to-contribute).

#### Disaster recovery

The 3 data centers should be involved in the smart city platform, then in case of serious problem in the main and secondary data center, the third DC, more protected from disaster like flood, can take over, and keep the platform online.

The databases used within the platform, must be replicated in real time on the third data center, to be able to use it in the shortest time possible.

#### GDPR

Observing General Data Protection Regulation will be mandatory on May 25th 2018, it’s a recent European directive defining new rules for personal data management.

Here are the major changes that are mentioned in this new legislation:

* **Consent:** Consent of personal data must be freely given, specific, informed and unambiguous. Consent is not freely given if a person is unable to freely refuse consent without detriment.
* **Accountability and privacy by default:** The GDPR has placed great emphasis on the accountability for **data controllers** to demonstrate data compliance. They will be required to maintain certain documentation, conduct impact assessment reports for riskier processing and employ data protection practices by default, such as data minimization.
* **Notification of a data breach: Data controllers** must notify the Data Protection Authorities as quickly as possible, where applicable within 72 hours of the data breach discovery.
* **Sanctions:** This new legislation allows the Data Protection Authorities to impose higher fines – up to 4% of annual worldwide turnover. The maximum fines can be applied for discrepancies related to international data transfers or breach of processing principles, such as conditions for consent. Other violations can be fined up to 2% of annual worldwide turnover.
* **Role of data processors: Data processors** will now have direct obligations to implement technical and organisation measures to ensure data protection, this could include appointing a **Data Protection Officer** if needed.
* **One stop shop:** This legislation will be applicable in all EU states without the need of implementing national legislation. Having a single set of rules will benefit businesses as they will not need to comply with multiple authorities, streamlining the process and saving an estimate of €2.3 billion a year.
* **Removal of notification requirement: T**he requirement of notifying or seeking approval from a Data Protection Authority is going to be removed in many circumstances. This decision is made to save funds and time. Instead of notification the new directive requires **data controllers** to put in place appropriate practices for large scale processing in the form of new technology.
* **Right to be forgotten:** Persons will be able to require their data to be deleted when there is no legitimate reason for an organization to retain it. Following, the organisation must also take appropriate steps to inform any third party that might have any links or copies of the data and request them to delete it.
* **Expanded territorial reach:** Companies that are based outside of the EU, but targeting customers that are in the EU will be subject to the GDPR which is not the case now. (Not Applicable for the project).

Data Protection Act identifies Data controllers and Data Processors as separate jobs:

<https://ico.org.uk/media/for-organisations/documents/1546/data-controllers-and-data-processors-dp-guidance.pdf>

The new GDPR rules described above will have to be applied mostly in the applications involving the use of personal data (in particular data from citizens).

However, as the platform will be recipient of many different kind of data, it could be hard to embrace the complexity of the data sets possible entanglement, some data can be related to citizens, because coming from objects/sensors at home, even weared, or simply published through applications.

Then to have a Data Protection office is worth to consider (if not already acted), but this is beyond the scope of this document.

The multitenant nature of the Fiware platform help a lot to data protection & privacy by dividing into independent services with secure and controlled (authorized and authenticated) access to data-sets, provided by global security IDM component.

However, in the case of cross usage of multiple history datasets like in some data-science cases, data must be anonymized and minimized, before being delivered for analysis, this is to consider either at data acquisition step and data consuming step.

## System Architecture Component Definitions

### Docker Images Management Components

MNCA needs to manage its own Docker images in a local Docker Images Registry, [Portus](http://port.us.org/) has been chosen as the component for this role.

It’s not a Fiware functional module, but an infrastructure element to manage a Docker registry (Docker images storage).

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql or MySQL (default) => Citus/PostgreSQL |
| Application | Docker Registry  NGINXor[**Puma**](http://puma.io/)(Ruby web server)  Portus web UI (developed in Ruby) |

Portus provides mainly authentication towards a Docker registry, which have to be started in a container: [Docker documentation](https://docs.docker.com/registry/deploying/#copy-an-image-from-docker-hub-to-your-registry).

Portus and the docker registry will be deployed as services, through a Docker compose or stack file: [example of secure compose file](https://github.com/SUSE/Portus/blob/master/examples/compose/docker-compose.yml)

There’s no real need of scalability for this component, only fault tolerance and high availability are required.

It should be installed on each Swarm manager node.

### Docker Swarm Management Component

Swarm manager is the orchestrator of the services based on the components of the platform.

It’s part of Docker software suite.

Swarm is cluster management integrated with Docker Engine (appeared in version 1.12)

To insure high availability several Docker nodes should be swarm manager, at the minimum one in each of the 3 DCs, there’s no real need of scalability, since it’s an internal orchestration management interface.

Even the choice is to setup the platform using Docker Swarm via Docker CLI, it could be valuable to use a container management solution such as [Rancher](http://rancher.com/)

Rancher is a complete container management open-source platform that makes managing and using containers in production really easy.

It allows to deploy services stacks over several Docker orchestration tools such as: Kubernetes, Swarm, Mesos.

It allows adding hosts directly from cloud providers or adding a host that’s already been provisioned. For cloud providers, host are provisioned using docker-machine and support any images that docker-machine supports.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MySQL (default) |
| Applications | Rancher server  Rancher UI  all in one Docker container |

### Security Component IDM GE

This component is the Fiware GE responsible for Fiware platform global user identity & access rights management

It is composed of 3 elements:

* the Identity Manager GE, in fiware catalog: [Keyrock](https://catalogue.fiware.org/enablers/identity-management-keyrock)
* the Access Control GE, in fiware catalog: [AuthZForce](https://catalogue.fiware.org/enablers/authorization-pdp-authzforce) (works with PEP Proxy)
* GE module to add security to backend applications, in fiware catalog: [PEP Proxy](https://catalogue.fiware.org/enablers/pep-proxy-wilma)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | changed to Postgresql/Citus cluster instead of MySQL |
| Applications | Keyrock (Keystone API backend + front-end: Horizon Django based application)  AuthZforce  PEP Proxy |

The security component is central to the platform, and so, used by most of other components from the platform to provide them authentication and authorization management, such as Orion Context Broker, Wirecloud, CKAN, CEP, Spagobi/Knowage, as well as Business ecosystem.

The Keystone IDM, back-end of Keyrock, can easily connect to an external LDAP, to authenticate and authorize users registered in it, new users can still be added and managed through Keyrock, but not registered in the LDAP (read only), once a user has been identified through LDAP, the access rights are managed through the security component.

AuthZForce is the PDP ((Policy Decision Point), PEP Proxy the PEP (Policy Enforcement Point).

Applications will be deployed through a service (in a Docker Stack file) providing high-availability and easy scalability, and will use the PostgreSQL/Citus cluster for data persistence.

### Orion Context Broker Component

This component is central to the Fiware platform: this is the data exchange bus of the platform.

The reference implementation of the Publish/Subscribe Context Broker GE.

Orion Context Broker allows to manage the entire lifecycle of context information including updates, queries, registrations and subscriptions.

It is an [NGSIv2](http://fiware.github.io/context.Orion/api/v2/stable/) server implementation to manage context information and its availability.

On Fiware catalog: [Orion Context Broker](https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker)

On Readthedocs: [Orion documentation](https://fiware-orion.readthedocs.io/en/master/)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MongoDB |
| Applications | Orion context broker  Rush notification relayer (queue notifications to reduce load on Orion) |

Context information follows the model presented in figure 5:

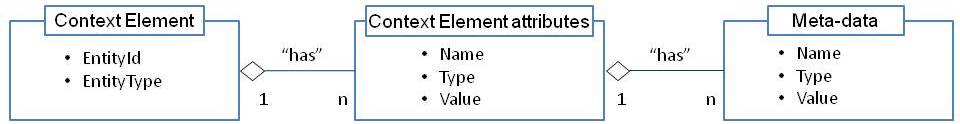


Figure 7: Context Element Model

A context element must provide an Entity ID, and an Entity Type, it can have as many attributes as needed, and each attribute can have as many Meta-data as needed.

The NGSI RESTful API manipulate context elements in JSON format.

Context Broker as a central component interact with many other components such as IOT Agents, Cosmos (Cygnus and Comet), CEP, Wirecloud

Context Broker is secured through the Keyrock IDM security component and PEP proxy.

To provide high availability and scalability, the Orion component is deployed as a service through a Docker stack file, meaning that several container instances of Orion will run to provide HA, scalability.

To reduce the load on Orion for managing notifications it’s possible to use a specific additional component: [Rush](https://fiware-orion.readthedocs.io/en/master/admin/rush/index.html) as a notification relayer ([Rush on Telefonica github](https://github.com/telefonicaid/Rush)), a Docker file: [OrionRush on github](https://github.com/Geonovum/sospilot/tree/master/src/fiware/docker/orionrush)

### MongoDB Cluster Component

This component is the [MongoDB](https://www.mongodb.com/) NoSQL database cluster global to the platform.

Several Fiware components use MongoDB to persist data, among them: Orion, IDAS, Comet STH.

Clustering is native in MongoDB, a MongoDB cluster can be scaled horizontally, sharding can be done on documents through replicas.

The component is deployed as a cluster of MongoDB nodes as containers in the swarm, as a Docker service through a Stack file.

Each container should point to a volume on the host for data persistence.

### Citus/PostgreSQL Cluster Component

This component is the cluster of [PostgreSQL](https://www.postgresql.org/) database servers of the Fiware platform.

The component is based on [Citus](https://www.citusdata.com/product/community), an open source clustering solution for PostgreSQL.

PostgreSQL is used by several components in the platform: such as CKAN, Wirecloud, and those that will be changed from MySQL to PostgreSQL like Keyrock, Biz Eco System.

Citus extends PostgreSQL to support distributed SQL queries. On top of PostgreSQL, Citus comes with its own transparent sharding, replication, distributed query planner and executor logic which enable execution of distributed SQL queries in parallel.

This provides Hadoop-like fault tolerance, scalability and recovery from mid-query failures while allowing large datasets to be queried orders of magnitude faster than what has been possible on PostgreSQL before.

In Citus architecture, the manager is a specific PostgreSQL database server instance that needs a replication to insure manager fail-over and provides complete high availability.

Several replication solutions exist for PostgreSQL, the recommendation from CitusData is [PostgreSQL integrated Streaming replication](https://www.postgresql.org/docs/10/static/warm-standby.html).

### Galera/MySQL Cluster Component

This component should be the cluster of MySQL database servers of the Fiware platform.

**This component will be installed only if changing from mySQL to PostgreSQL for a module is too complex or impossible.**

Any MySQL database that can’t be replaced with PostgreSQL, should be replaced by MariaDB,

To provide high availability and scalability the component will be based on a MySQL/MariaDB clustering solution: [Galera cluster](http://galeracluster.com/products/).

There is a MariaDB and Galera integration chapter on [MariaDB web site](https://mariadb.com/kb/en/library/galera-cluster/).

### Cosmos Big Data Component

This component is the Big Data and data history GE of the Fiware platform.

As the most complex component, it’s composed of many modules:

* Cygnus, as an interface between the context broker and the different data history storages
* Comet STH for Short Term History storage, uses MongoDB to persist the data sent through the context broker.
* An Hadoop ecosystem (HAAS) to store and process (batch or stream) long term history for analytics purposes.

Hadoop HDFS provides Distributed storage.

Distributed batch processing (with MapReduce).

Distributed stream processing through Sinfonier (Apache Storm based) or Spark.

Also, almost any software of the Hadoop ecosystem can be used in Cosmos: such as Hive, Pig, zookeeper, yarn, HBase…

On Fiware catalog: [Cosmos](https://catalogue.fiware.org/enablers/bigdata-analysis-cosmos)

On readthedocs: [Cosmos documentation](http://fiware-cosmos.readthedocs.io/en/latest/)

On Fiware Forge: [Architecture documentation](http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.BigData)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MongoDB cluster |
| Applications | Cygnus  Comet STH  Hadoop ecosystem: Sahara HAAS engine or HAAS light version engine  Tidoop (MapReduce)  Sinfonier (stream analytics Apache storm based) |

### Knowage Analytics Component

This component is for business intelligence, Fiware have originally chosen SpagoBI as the reference platform for this Analytics GE.

In the Fiware catalog is proposed the evolution of SpagoBI to a more integrated suite with all modules included Knowage

On Fiware catalog: [Knowage](https://catalogue.fiware.org/enablers/data-visualization-knowage)

On readthedocs: [Knowage in Fiware](http://knowage.readthedocs.io/en/latest/)

Knowage is a professional open source suite for modern business analytics over traditional sources and big data systems.

This GE provides the following features:

* BD (big data), to analyse data stored on big data clusters or NoSQL databases
* SI (smart intelligence), the usual business intelligence on structured data, but more oriented to self-service capabilities and agile prototyping
* ER (enterprise reporting), to produce and distribute static reports
* LI (location intelligence), to relate business data with spatial or geographical information
* PM (performance management), to manage KPIs and organize scorecards
* PA (predictive analysis), for more advanced analyses
* EI (embedded intelligence), to link Knowage with external solutions provided by the customer or third parties.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | PostgreSQL/Citus for metadata storage |
| Applications | A set of java modules deployed under a Tomcat |

The composer part of Knowage, shouldn’t be exposed to public use, if some dashboards or reports visualization needs to be published, it need to be integrated in applications in an iframe component or by using the API to get the Knowage generated content.

### CKAN Open Data Component

This component is the chosen Fiware GE to manage Open Data.

CKAN GE at Fiware: <https://catalogue.fiware.org/enablers/ckan>

Fiware Docker installation: <https://github.com/okfn/docker-fiware-ckan>

Official installation guide : <http://docs.ckan.org/en/ckan-2.3.5/maintaining/installing/install-using-docker.html>

The CKAN GE provides the following features:

* Complete catalog system with easy to use web interface and a powerful API
* Strong integration with third-party CMS’s like **Drupal** (currently in use at MNCA) and WordPress
* Data visualization and analytics
* Workflow support lets departments or groups manage their own data publishing
* Fine-grained access control
* Integrated data storage and full data API
* Federated structure: easily set up new instances with common search

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus cluster  Apache Solr (for search feature) |
| Applications | Api and WEB UI |

### IDAS IOT Components

The IDAS component is an implementation of the Backend Device Management GE.

It provides software for IOT use cases, like:

* to manage communication between sensors & actuators/devices/gateways and the data context broker using NGSI REST API.
* to manage provisioning of devices through a REST API, it is composed of several components organized by south bound protocols (UL2, JSON, LWM2M, Sigfox)

On Fiware Catalog: [IDAS](https://catalogue.fiware.org/enablers/backend-device-management-idas)

Documentation on the Fiware IOT stack : [by Telefonica](https://fiware-iot-stack.readthedocs.io/en/latest/)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MongoDB |
| Applications | [UL2.0 IOT Agent](https://github.com/telefonicaid/iotagent-ul)  [JSON IOT Agent](https://github.com/telefonicaid/iotagent-json)  [OMA LWM2M IOT Agent](https://github.com/telefonicaid/lightweightm2m-iotagent)  [SIGFOX IOT Agent](https://github.com/telefonicaid/sigfox-iotagent)  [IOT Agent library to develop other agents](https://github.com/telefonicaid/iotagent-node-lib)  IOT Agent Manager (under upgrade for R5 release)  IoT Edge Manager (forthcoming for R5)  [IOT Orchestrator with portal UI](https://github.com/telefonicaid/orchestrator) (from Telefonica)  MQTT broker like [Mosquitto](http://mosquitto.org/) is needed for MQTT connectivity of IOT agents.  [Node-RED](https://nodered.org/) should be used to provide pre-processing for some IOT data sources. |

As the IOT agent components provide only a REST API to interact with, it lacks a web user interface for device management in the current Fiware catalog.

The [IOT Agent Manager](https://github.com/telefonicaid/iotagent-manager) is under porting in Nodejs for release 5, it should provide an IOT agents management layer.

Most of the IOT modules in Fiware catalog come from [Telefonica, in their github](https://github.com/telefonicaid), they provide a component called [Orchestrator](https://github.com/telefonicaid/orchestrator), that allows to interact with the different modules: IOT agents, Perseo CEP, Orion, IDM, this orchestrator component also provide a web portal ready to use, but not open-source.

An IOT agent for [OPC UA](https://en.wikipedia.org/wiki/OPC_Unified_Architecture) is under development: <https://github.com/BEinCPPS/idas-opcua-agent> it should allow to exchange information between an OPC server and an NGSI context broker.

After analysis of already used data sources, it appears that none is immediately compatible with the proposed IOT agents, then some adapters need to be developed, to be able to store it in Fiware, the data coming from those different data sources.

The existing data sources are most of time files in different format (XML, CSV, formatted text), coming through FTP, or got through HTTP

Those data need to be converted to JSON in a first step (with Node-RED), and posted to an MQTT broker, then a **JSON IOT Agent** subscriber to the topic on the MQTT broker, should convert into NGSI and post to the context broker.

The first step (data acquisition, conversion to JSON and publish to MQTT) will be done through specific programs or scripts, one per type of data source.

The components will be deployed as docker services through stack files.

### IOT Broker Component

The IOT Broker GE is a lightweight and scalable middleware component that separates IoT applications from the underlying device installations.

This component sits between the Context broker, which manipulate things context, and the IOT agents which manipulate device data, and it allows to define associations between device-level and thing-level information, and to filter information before sending it to the context broker.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | CouchDB for historical agent  PostgreSQL (Citus cluster) for utility storage |
| Applications | IOT Broker |

This component is not mandatory for the initial setup of the smart city platform, a reflection is needed to evaluate the opportunity for its inclusion in the target platform.

### IOT Discovery Component

The IOT Dicovery GE role is to act as a meeting point for

* IoT Context Producers to register the availability of their Things and Sensor devices,
* IoT Context Consumers to discover them.

This using either the OMA NGSI-9 messaging protocol, API for contextual information exchange.

For semantic users, the Sense2Web API can be used which supports Linked Open Data.

The API exposes two main modules:

* NGSI-9 Server which provides a repository for the storage of NGSI entities and allows NGSI-9 clients (Data Handling GE, Device Management GE, IoT Broker) to:
  + Register context information about Sensors and Things.
  + Discover context information using ID, attribute, attribute domain, and entity type.
* Sense2Web Linked-data platform which provides a semantic repository for IoT providers to register and manage semantic descriptions (in RDF/OWL) about their "Things": Sensor/Actuator Devices, virtual computational elements (e.g. data aggregators) or virtual representations of any Physical Entity.

Sens2Web provides IoT users to discover these registered IoT elements using:

* Descriptions in RDF
* A probabilistic search mechanism that provides recommended and ranked search results for queries that don’t provide exact matching property values.
* Semantic querying via SPARQL
* An association mechanism that associates Things and sensors based on their shared attribute (e.g. temperature) and spatial proximity, which can then be queried via SPARQL.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MySQL default to replace by PostgreSQL (Citus cluster) |
| Applications | NGSI-9 server (Tomcat application)  Sense2Web Linked-data platform (Tomcat application) |

This component is not mandatory for the initial setup of the smart city platform, a reflection is needed to evaluate the opportunity for its inclusion in the target platform.

### CEP Component

The CEP component is the Complex Event Processing GE for the Fiware platform.

It is a scalable integrated module to support the development, deployment, and maintenance of event-driven applications.

To summarize: through the CEP component it’s possible to configure rules on incoming data (like: if a fridge temperature is higher than 4°C),

that if verified, generate an event/action (an alarm is sent, mail, SMS, or displayed on a cockpit/dashboard)

In Fiware catalog is proposed an implementation from IBM of the CEP module: [PROTON](https://catalogue.fiware.org/enablers/complex-event-processing-cep-proactive-technology-online)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | None in this version |
| Applications | Proton is based on Tomcat and uses 4 wars:   * ProtonOnWebServerAdmin * ProtonOnWebServer * AuthoringTool * AuthoringToolWebServer |

Another CEP component is provided by Telefonica: [Perseo](https://github.com/telefonicaid?utf8=%E2%9C%93&q=perseo&type=&language=)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | MongoDB |
| Applications | [Perseo Core](https://github.com/telefonicaid/perseo-core)  [Perseo FE](https://github.com/telefonicaid/perseo-fe) |

After testing of both solutions, it appears that Perseo is simpler and more straight forward to use than Proton, **so Perseo should be the chosen implementation**.

The CEP component will be deployed as a Docker service, using stack file.

### Application Mashup Components

Wirecloud is the Fiware GE to build Rich Internet Applications (RIA), using semantic technologies to offer a next-generation end-user centered web application mashup platform.

On Fiware Catalog: [Wirecloud](https://catalogue.fiware.org/enablers/application-mashup-wirecloud)

On Readthedocs : [Wirecloud Documentation](https://wirecloud.readthedocs.io/en/stable/)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus |
| Applications | Wirecloud web application  Wirecloud back-end server |

Wirecloud is a web application mashup platform aimed at allowing end users without programming skills to easily create web applications and dashboards/cockpits.

It integrates completely to Fiware architecture, by connecting directly to Context Broker and historic data storage components such as Comet STH.

Mashups and Dashboards made with Wirecloud, can be monetized through the business ecosystem component

Alternatively a small javascript library like [Freeboard](https://github.com/Freeboard/freeboard) can be used to visualize data coming from NGSI context broker: [news on Fiware](https://www.fiware.org/2015/07/13/fiware-orion-data-source-now-available-for-freeboard/). (an [Orion FIWARE datasource](https://github.com/telefonicaid/fiware-connectors/tree/master/FreeBoard-Orion-Plugin) has been developed by Telefonica)

Freeboard can be used also from web where you can publish dashboards: [Freeboard.io](https://freeboard.io/)

It can also be deployed on a web server, on the platform, but there’s no security or publishing management provided, it has to be added, by passing through a proxy.

As it’s pure javascript library and some json definition and configuration, it can be integrated in a web application that is proposed in the market place.

### Business Ecosystem Component

The Business API Ecosystem exposes its complete functionality through TM Forum standard APIs; concretely, it includes the following key features:

* Support for the management of catalogs, products, and offering
* Support for rich pricing models, including recurring payments, pay-per-use, etc.
* Support for accounting callbacks
* Support for billing and charging
* Integrated support for PayPal, including customer charges and seller payments
* Support for revenue sharing, including models with multiple stakeholders involved

Link to Fiware catalog: [Business Api Ecosystem](https://catalogue.fiware.org/enablers/business-api-ecosystem-biz-ecosystem-ri)

Link to the documentation: [on](http://business-api-ecosystem.readthedocs.io/en/latest/) readthedocs

This component is the **Market Place** of the Smart City platform, it allows to leverage:

Authorization access to services, and their monetization, even with several stakeholders involved that share revenues,

services here includes the applications available in the catalog, the data sets in CKAN, the available APIs.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus (to change from MySQL) |
| Applications | Web application  back-end servers:   * TM Forum APIs reference implementation (catalog, ordering, inventory, usage, billing, party Apis) * Charging Backend, * Revenue Sharing Service (RSS) * Logic Proxy |

The components will be deployed as services in containers, through a stack file.

### Development Components

Fiware was proposing to use [FusionForge](https://fusionforge.org/) (the software suite originated from Forge that was running SourceForge.net), Eclipse as an IDE as the set of tools for development.

Now with the advent of Docker, Eclipse have evolved towards the container paradigm and propose [Eclipse Che](https://www.eclipse.org/che/) as an IDE & Developer Workspace server that works with Docker.

**It allows to develop Docker ready applications**, in addition it provides a RESTful API to interact with.

If for many applications like dashboards, using Wirecloud component is a good solution, for more complex or even mobile application it is not adapted, so that Eclipse Che IDE is an appropriate solution to develop Docker ready applications.

Eclipse Che is the next generation of IDE, it provides integrated connectors to git and SVN, so that it can be easily connected to the Gitlab of MNCA.

An Eclipse Che server can be set in a multi-users environment, it uses [Keycloak](http://www.keycloak.org/) as IDM provider, and a PostgreSQL database for metadata.

Keycloak comes also with a built-in LDAP/AD plugin, which supports password validation via LDAP/AD protocols and different user metadata synchronization modes, so that internal users already registered on MNCA ADAM, can be identified in Eclipse Che.

An Eclipse Che Server using the IDM component, is the right candidate for the development IDE for the MNCA smart city platform, as the component works natively with Docker, it will be part of the PAAS provided in the platform.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus |
| Applications | Elipse Che  Keycloack (or Keyrock) for security  GitLab SCM/PM |

Deployed as a container, the component can be replicated to provide HA, and scaled up, by using a service through the use of Docker stack file.

### Monitoring Components

[Portainer](https://portainer.io/) will be used for real time management & monitoring of services and the composing containers of the swarm.

Portainer on [Github](https://github.com/portainer/portainer/blob/develop/README.md).

Portainer will be replicated on several nodes with several instances, ideally on each swarm manager (mandatory to have vision of the swarm).

As Portainer don’t use a Database, but store it’s configuration in files, mainly for user and dependencies management, so that those configuration files should be stored in a secured NFS storage volume, shared by all instances.

Fiware in its catalog provides modules intended for monitoring: [on Telefonica github](https://github.com/telefonicaid?utf8=%E2%9C%93&q=Monitoring&type=&language=), it’s composed of some probes sending information about the status of a component, to a module which gather and convert the information, before sending it to a monitoring framework (Nagios, Zabbix…)

[Centreon](https://www.centreon.com/) is already in use as the monitoring framework of MNCA.

containers should be instrumented with needed check modules to monitor services and nodes in the platform from the Centreon, an additional Centreon service could be added in the platform if needed.

One thing to think about, is the expansion of the platform, in terms of numbers of services deployed, this will lead to complexify the task of monitoring configuration, if it remains manual, so that it’s worth to think about automatic services discovery to simplify and automate monitoring for OPS

One solution should be to use [Consul](https://www.consul.io/) from Hashicorp for services discovery, but Centreon provides also a module for [auto discovery of services](https://documentation.centreon.com/docs/centreon-auto-discovery/en/latest/) (commercial) or a free community version: [centreon-discovery](https://github.com/Centreon-Community/centreon-discovery)

[Rancher](http://rancher.com/) provides monitoring on containers and nodes, and in swarm mode, it deploys also a Portainer.

Another concern is to be able to access any of the log files produced by components, [Docker supports many different logging drivers](https://docs.docker.com/engine/admin/logging/overview/#supported-logging-drivers): default is json-file, but Journald or syslog can be used.

An interesting option should be to use [gelf logging driver](https://docs.docker.com/engine/admin/logging/gelf/), which writes log messages to a Graylog Extended Log Format (GELF) endpoint such as Graylog or Logstash, using Logstash the logs can be pushed into an ElasticSearch instance for analysis.

### Poi Data Provider Component

Fiware GE to provide Point Of Interest (POI) management.

On Fiware Catalog: [POI Data Provider](https://catalogue.fiware.org/enablers/poi-data-provider)

POI (Points of interest) Generic Enabler is a web server kit that supports

* storing information related to locations
* serving queries by location and other criteria
* can be configured to meet your data needs

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus + [PostGIS](http://postgis.net/) (spatial extension)  MongoDB |
| Applications | API back-end server written in PHP |

This component is not necessary to the platform, since POIs should already be represented as entities in the context broker.

Nevertheless, in some use cases such kind of component can be useful, in particular for location based applications with high demand for POIs.

### GIS Data Provider Component

Fiware GE to provide 3D spatial data storage.

This GE is able to host geographical data and serve it in 3D form (where applicable) to both mobile and web clients. The GE implementation is based on open source Geoserver project (GPL licenced) and [W3DS](http://www.w3ds.org/doku.php) (Web 3D Service) extension.

To note that W3DS has been replaced at 3DPS ([3D Portrayal Service](http://docs.opengeospatial.org/is/15-001r4/15-001r4.html)) at [OGC](http://www.opengeospatial.org/), and moreover the Geoserver W3DS extension doesn’t seem to exist anymore in recent versions.

On Fiware catalog: [GIS data Provider](https://catalogue.fiware.org/enablers/gis-data-provider-geoserver3d)

On readthedocs: [GIS Data Provider](http://gisdataprovider.readthedocs.io/en/latest/index.html)

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | Postgresql/Citus + [PostGIS](http://postgis.net/) (spatial extension) |
| Applications | [GeoServer](http://geoserver.org/) |

This component can be installed as a Docker container: one of most used image on [DockerHub](https://hub.docker.com/r/neowaylabs/geoserver/)

The one from Fiware is on [DockerHub](https://hub.docker.com/r/fiware/gisdataprovider/), and provides the W3DS extension.

This GIS component is provided to be able to store 3D spatial entities (in [XML3D](http://xml3d.org/)), and use it through the W3DS interface in AR or VR applications, it has no special connection with other Fiware components.

As soon as the needs are not identified for some application, it’s not really worth to install this component.

In addition, if the installation is very simple, the data must be prepared and inserted in the spatial database behind the Geoserver.

The MNCA GIS department already uses ESRI products like ArcGIS to work on spatial data, they might produce XML3D models to populate the GIS data provider.

### Load Balancing Component

This component is in charge of load balancing and dispatching inbound traffic from outside to all internal components that need to provide an interface opened to public access.

Initially it was envisaged to use an F5 BigIP appliance, but the alternative using [HAProxy](http://www.haproxy.org/) has been explored, and the choice is to start with this open source solution.

|  |  |
| --- | --- |
| **Architecture Component** | **Component Elements** |
| Database Server | none |
| Applications | [HaProxy](http://www.haproxy.org/) |

To separate the concerns between the southbound IOT part and the northbound application part, each bound will have its own HaProxy, each of those should be deployed as Docker service, to provide high availability (fail over redundancy).

# Section 3 SYSTEM ARCHITECTURE DESIGN

*The* ***System Architecture Design*** *section provides detailed descriptions of each product implementing architecture components, and explains the rationale for product selection.*

*For each* ***System Architecture Component*** *(identified in Section 2.3 above), this section describes: specific* ***Component Functions****, requirements and other* ***Technical Considerations*** *that were used in the decision-making process, as well as any specific* ***Products*** *selected to implement this component. The* ***Selection Rationale*** *identifies any other products that may have been considered, and provides rationale for the decision.* ***Architecture Risks*** *identifies any potential risks associated with the architecture element and the solutions to minimize those risks.*

## Docker Images Management Component

### Component Functions

This component is not a Fiware functional component, it is used to manage the access to Docker Images used to deploy the smart city platform in production, in a Docker Images Registry, generally private.

It is a manager over an installed Docker registry, that is used mainly to authenticate users to authorize them to manipulate Docker Images files from the attached Registry.

A Docker registry is a repository of built Docker images of containers (DockerHub is a public Docker registry).

Registry management is needed in production, but also in development and QA steps of a project

If Portus is the choice for production container images management,

GitLab is the container images management to use while development and QA.

### Technical Considerations

Portus an [open source software originated from OpenSuse](https://github.com/SUSE/Portus), written in Ruby.

It provides its own security access layer that can be configured to use an LDAP server for the authentication, then it is possible to configure the component to use MNCA MS ADAM identity management.

At least one local Docker registry must be installed as a Docker container.

Portus component needs a database, MySQL is proposed by default, PostgreSQL is also supported.

The database is needed for user management.

Portus can be configured to use an LDAP server for the authentication: [Portus LDAP support](http://port.us.org/features/2_LDAP-support.html), so it can be interfaced with the MNCA ADAM.

Portus and the Docker registry will be deployed as services, through a Docker compose or stack file: [example of secure compose file](https://github.com/SUSE/Portus/blob/master/examples/compose/docker-compose.yml)

Incoming port: 3000/TCP

### Selected Product(s)

The selected product to manage images in docker registry is Portus, but the registry itself is provided through a Docker registry container, so that Portus mainly provides authentication to allow access to the images

GitLab provides also Docker registry management for container images: [GitLab Container Registry](http://docs.gitlab.com/ce/user/project/container_registry.html).

It allows the management of Docker container images in Gitlab this could be used in development and QA (testing), and keep Portus and the associated registry as the registry for production.

### Selection Rationale

Several alternatives exist for Docker Images management UI, Portus is a well-known open-source distribution provided by [OpenSuse](https://www.opensuse.org/).

Portus has already been adopted during MNCA Fiware test by Olivier Sevilla, summer 2017.

Nevertheless, for development and QA, the use of specific registry attached behind the Gitlab of MNCA will allow to manage lifecycle of the Docker images, before releasing it in production.

GitLab like Portus is not a repository itself, it is a registry manager,

### Architecture Risks

The MNCA Gitlab should receive the Docker images in its own registry for development and QA ([just need to [install if not] activate it](https://docs.gitlab.com/ee/administration/container_registry.html#enable-the-container-registry))

The registry for production should be managed with Portus

## Docker Swarm Manager Component

### Component Functions

This is not a real component, it’s an artifact included in Docker (since 1.12 version), which allows to manage a Docker Swarm.

Docker Swarm turns a group of Docker engines into a single virtual Docker engine, on which containers are running as services.

Dock Swarm is designed to work around four key principles:

* Simple yet powerful with a “just works” user experience
* Resilient zero single-point-of-failure architecture
* Secure by default with automatically generated certificates
* Backward compatibility with existing components

The very few simple manager API functions are: initializing a Swarm, add a machine/node to a swarm, remove a node from a swarm, and renew certificates.

Regarding network, Docker swarm generates two different kinds of traffic:

* **Control and management plane traffic:** This includes swarm management messages, such as requests to join or leave the swarm. This traffic is always encrypted.
* **Application data plane traffic:** This includes container traffic and traffic to and from external clients.

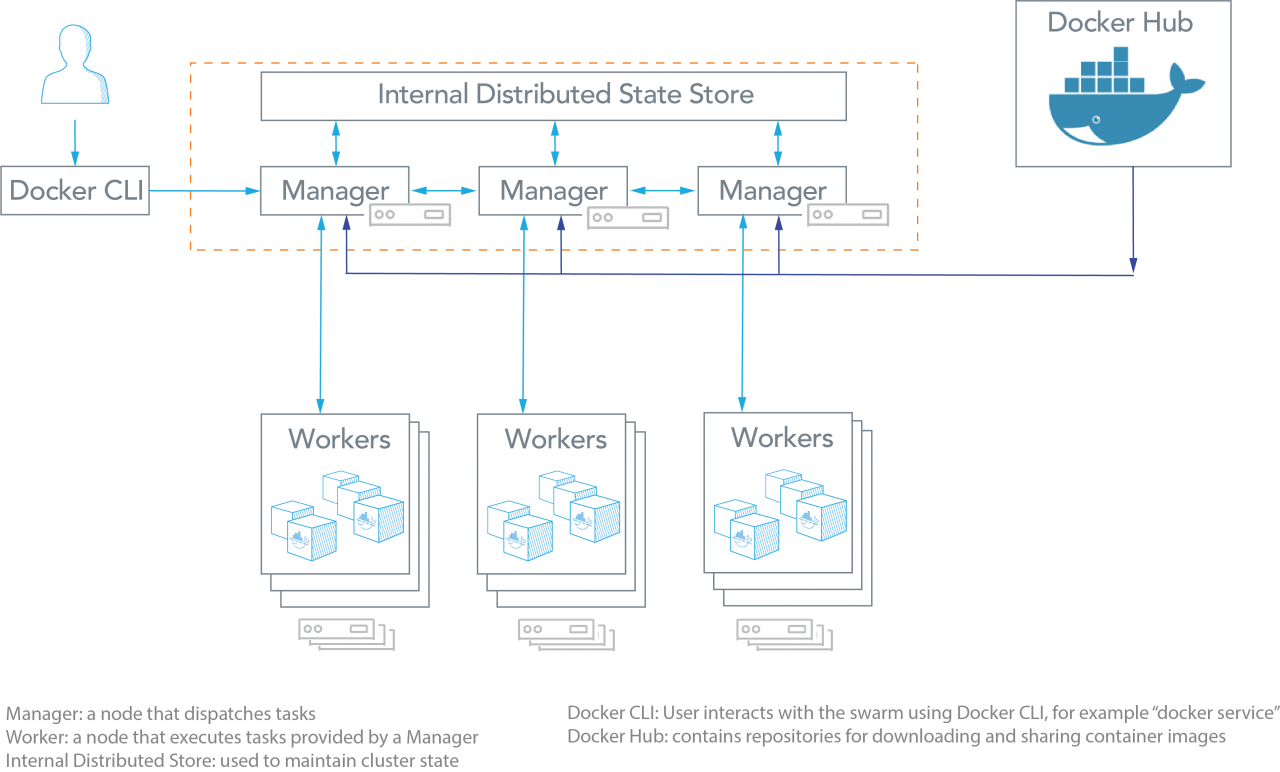


Figure 8: Docker Swarm Architecture

### Technical Considerations

Docker daemons participating in a swarm need the ability to communicate with each other over the following ports:

* Port 7946 TCP/UDP for container network discovery.
* Port 4789 UDP for the container **overlay** network.

This means that only those 2 ports need to be open in firewalls that sit between virtual networks on which the docker swarm is deployed.

The application data communication go through the Overlay network (TCP packets are encrypted and encapsulated in UDP packets)

### Selected Product(s)

In a first step Docker Swarm will be used to deploy the Smart City platform.

in a second step it should be interesting to install and use a container management solution such as [Rancher](http://rancher.com/)

Rancher is a complete open-source container management platform that makes managing and using containers in production really easy.

It allows to deploy services stacks over several Docker orchestration tools such as: Kubernetes, Swarm, Mesos.

It allows adding hosts directly from cloud providers or adding a host that’s already been provisioned. For cloud providers, host are provisioned using docker-machine and support any images that docker-machine supports, also it allows to manage several environments such as production, QA, dev.

In addition, it integrates natively some monitoring features.

### Selection Rationale

Docker Swarm is simple to learn compared to other solutions like Kubernetes or Mesos, this is the reason we’ll start with it to host the smart city platform.

A comparison between Docker swarm and Kubernetes explains well the differences: <https://platform9.com/blog/kubernetes-docker-swarm-compared/>

Adding Rancher in the game could help to simplify the management of the platform, and later replace Swarm with Kubernetes, to be able to scale automatically the platform.

### Architecture Risks

To achieve fault tolerance, several nodes should be set as Swarm managers, ideally one per data center.

It is advised to have an odd number of swarm managers (1=> SPOF, 3, 5….)

## Security Component IDM GE

### Component Functions

Manage globally the security for the Fiware platform, in terms of identity and access authorization to services and data.

The Keyrock module provides identity management Keystone back-end, the integrated web UI is provided through Horizon front-end, which allows user accounts management.

AuthZForce is the PDP (Policy Decision Point), PEP Proxy the PEP (Policy Enforcement Point).

The AuthZforce component provides roles and access rights management based on XACML

The PEP Proxy components provides Keyrock security to other backends of the platform such as Orion context broker.

The 3 components together offer 3 levels of security:

* Level 1: Authentication
* Level 2: Basic Authorization
* Level 3: Advanced Authorization

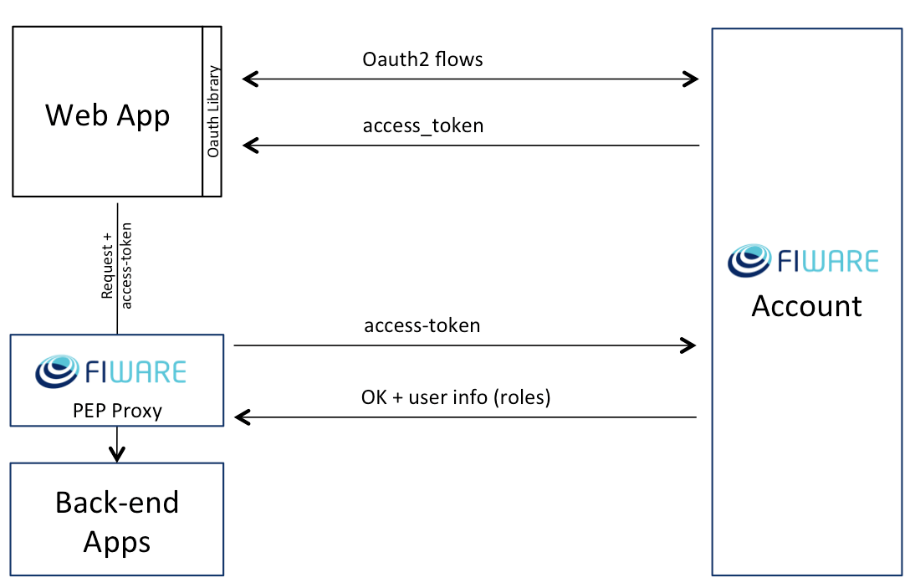


Figure 9: Level 1: Authentication only (Fiware Account is Keyrock IDM)

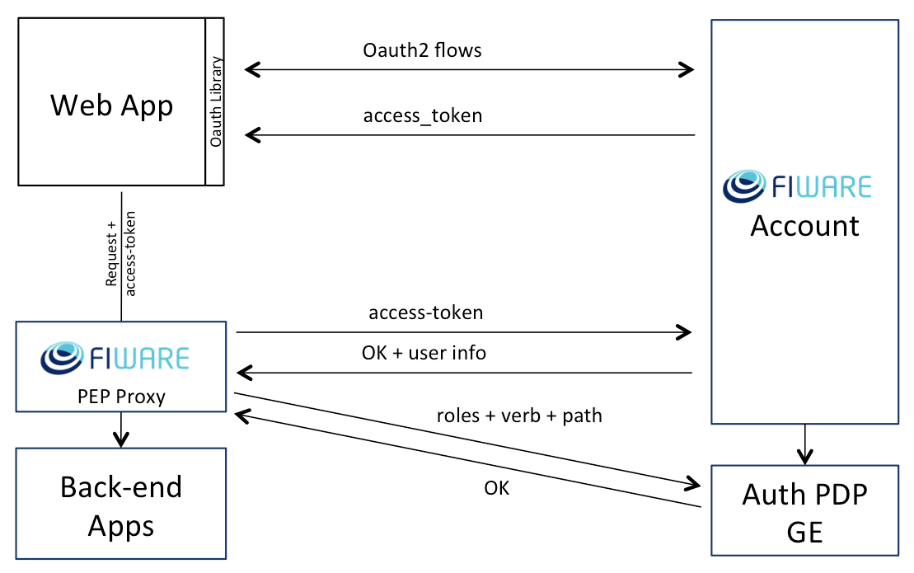


Figure 10: Level 2: Authentication & Basic Authorization

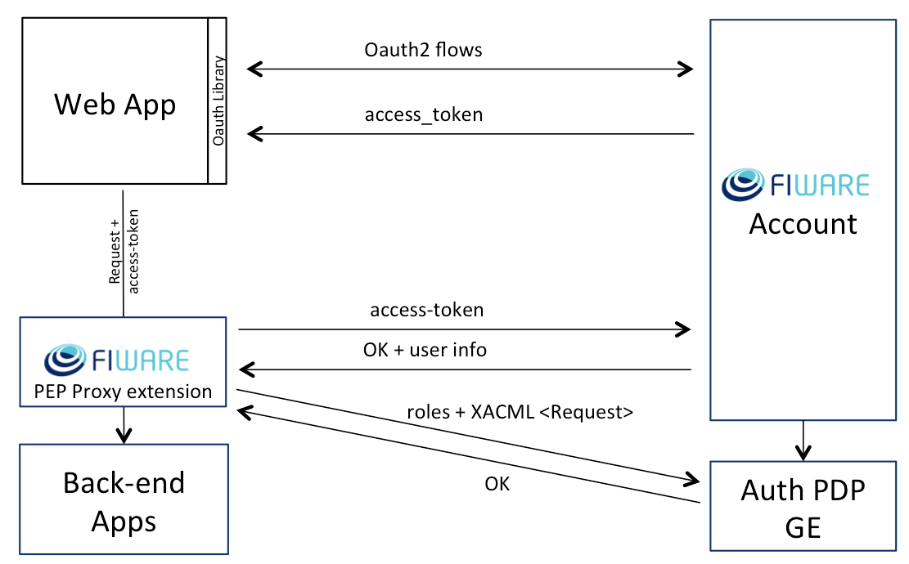


Figure 11: Level 3: Authentication & Advanced Authorization (using XACML).

### Technical Considerations

It’s based on **Keystone**, the IDM component of OpenStack cloud software suite.

it uses the standard OAuth2 protocol.

it can connect with an LDAP: [Integrate Identity with LDAP](https://docs.openstack.org/keystone/pike/admin/identity-integrate-with-ldap.html), the LDAP can be the MNCA ADAM: [Keystone with ADAM](https://wiki.openstack.org/wiki/HowtoIntegrateKeystonewithAD), just matter of configuration.

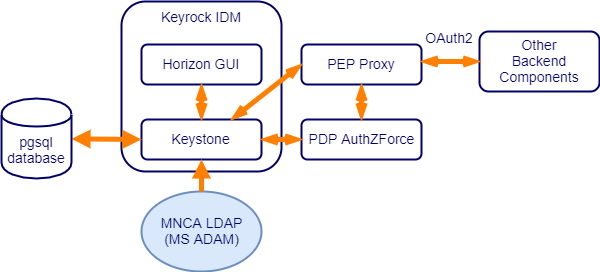


Figure 12: IDM components organization

Incoming ports:

* Keyrock API endpoint: 5000/TCP
* Horizon UI: 8000/TCP (in production 80 through NGINX)
* Keystone Admin endpoint: 35357/TCP
* PDP Authzforce:
  + 8080/TCP
  + 22/TCP

### Selected Product(s)

Keyrock identity manager is based on OpenStack **Keystone**

**Keystone** provides authentication, authorization and service discovery mechanisms via HTTP primarily for use by projects in the OpenStack family.

It is most commonly deployed as an HTTP interface to existing identity systems, such as LDAP.

### Selection Rationale

Keyrock, PEP Proxy, PDP AuthZforce are the components proposed in Fiware catalog to handle authentication and authorization, for any other component of the platform that support OAuth2.

Keyrock IDM GE is Based on OpenStack originated IDM component [**Keystone**](https://docs.openstack.org/keystone/latest/), which supports: Oauth2, OpenID connect, SAML and other protocols, OAuth2 is now widely used, Fiware mostly rely on it for authentication and authorization to use the services offered by the platform.

### Architecture Risks

As a globally used component for Identity, authentication and access rights management, it is a highly critical component.

The database must be maintained in a clustered environment that insure high availability as well as scalability, this is provided through the Citus cluster of PostgreSQL databases.

The involved application modules (Keyrock, AuthZforce, PEP Proxy), will be deployed as services through a docker stack file in the global swarm, it gives the possibility to specify the number of containers to run for each service, as the components are stateless, several containers can run together the same service, to provide High Availability and scalability.

## Orion Context Broker Component

### Component Functions

This component manages context information exchange for the whole platform using NGSI (V1 & V2) Api, this is a core feature of the Fiware Platform, as it provides the link between the data coming from things in the field, where data models are many, and their usage in the applications.

In addition to the query/answer paradigm, the component provides a publish/subscribe usage paradigm

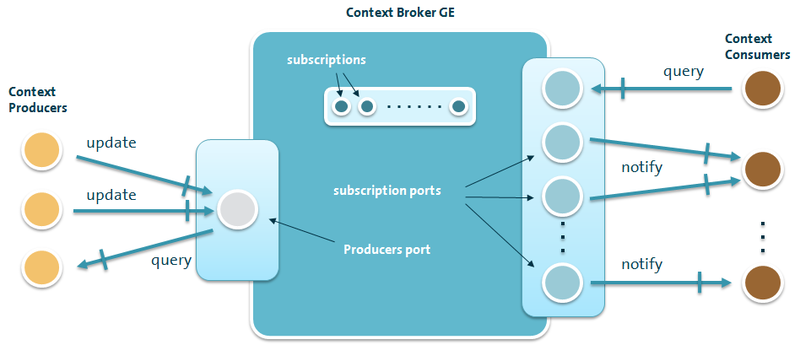


Figure 13: Context Broker interactions

### Technical Considerations

The Orion context broker is developed in C++

Orion doesn't provide "native" authentication nor any authorization mechanisms to enforce access control. However, authentication/authorization is provided using the [FIWARE PEP Proxy GE](https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Security.PEP_Proxy_Generic_Enabler).

Now, there are two GE implementantions (GEis) that can work with Orion Context Broker:

* [Wilma](http://catalogue.fiware.org/enablers/pep-proxy-wilma) (the GE reference implementation, is the chosen component for the platform)
* [Steelskin](https://github.com/telefonicaid/fiware-pep-steelskin)

Orion can be configured to use https for API access, and for notifications too: see documentation on [Readthedocs](https://fiware-orion.readthedocs.io/en/develop/user/security/index.html)

Orion Incoming API port: 1026/TCP

The RAM memory needed is minimum 4 GB, with the MongoDB, so the needed memory for Orion itself is less than that.

The size of the Orion container image is 260 MB.

The chapter “resource availability” give some hints about the needs on [Orion ReadTheDocs](https://fiware-orion.readthedocs.io/en/develop/admin/diagnosis/index.html#resource-availability)

The Orion component is easily deployable through a Docker service, how to do it is explained on the documentation chapter in [Orion ReatheDocs](https://fiware-orion.readthedocs.io/en/develop/user/docker/index.html)

The use of MongoDB version 3.x for data persistence is recommended (the MongoDB cluster will use the 3.6 version).

The Orion documentation on Readthedocs provides an entire chapter about the [performance tuning of the Orion context broker in production](https://fiware-orion.readthedocs.io/en/develop/admin/perf_tuning/index.html).

The following schema shows the place of Orion context broker in the Fiware architecture:

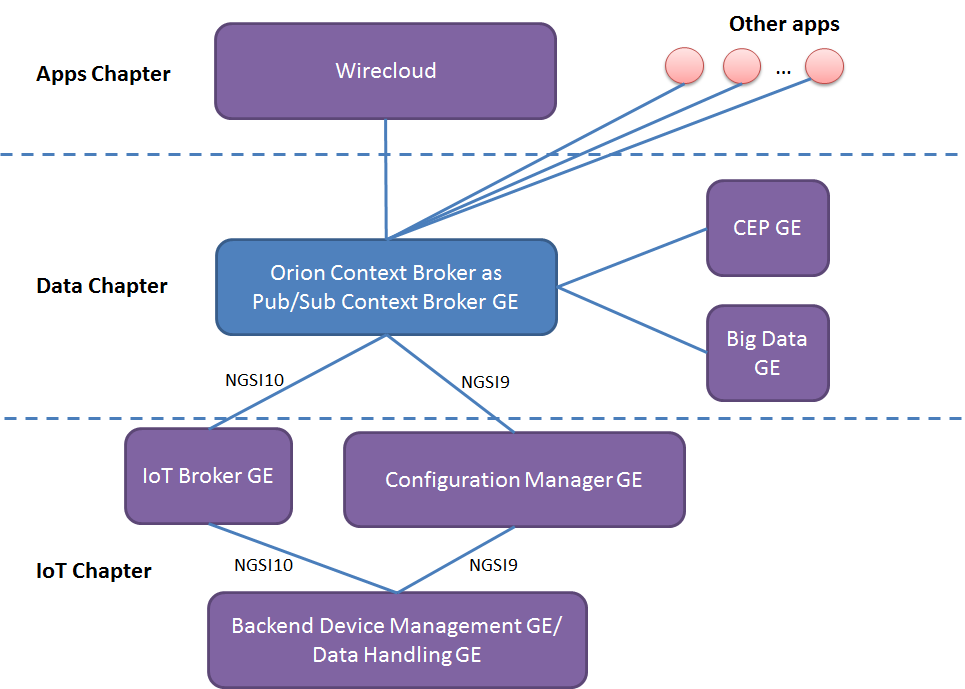


Figure 14: Orion Context Broker relation with other components

### Selected Product(s)

Orion Context broker is the reference GE in Fiware catalog.

There’s another implementation of an NGSI context broker originated from Orange:

But tailored for use at IOT gateway level, for example to manage a network of sensors relative to a specific activity among numerous smart city use cases

### Selection Rationale

Orion is the reference implementation of NGSI Context Broker in Fiware catalog.

Fiware is member of the ISG (Industry Specification Group) CIM (Context Information Management) created at ETSI beginning of 2017.

ISG CIM will focus on developing specifications for a common context information management API, data publication platforms and standard data models. The group will work closely with the **ETSI SmartM2M** technical committee and with [**oneM2M**](http://www.onem2m.org/), the global standards initiative for M2M and the IoT (Internet of Things) of which ETSI is a founding member, since the IoT is one of the sources of context data for smart applications.

The OMA NGSI information model is currently being evolved to better support linked data (entity relationships), property graphs and semantics (exploiting the capabilities offered by **JSON-LD**). This work is being conducted under the ETSI ISG CIM initiative.

It is noteworthy that the ETSI ISG CIM information model is a generalization of the existing FIWARE NGSI information model. As a result, it is expected a good level of compatibility and a clear migration path between both information models.

### Architecture Risks

This component is central to the platform, because it’s the dorsal of the Fiware architecture

It can become a bottleneck in case of high traffic, so a particular attention is needed on response time, through specific monitoring.

The MongoDB database used by the context broker is a global MongoDB cluster, which provide high availability and scalability.

The application component, Orion will be deployed as a service through a Docker stack file, to run several Orion containers, and will then provide high-availability and scalability.

you can also take advantage of Rush as notification relayer. Thus, instead of managing the notifications itself (including waiting for the HTTP timeout while the notification receives responses), Orion passes the notification to Rush, which in turn deals with it, to reduce the load on Orion.

[Rush](https://fiware-orion.readthedocs.io/en/master/admin/rush/index.html) as a notification relayer ([Rush on Telefonica github](https://github.com/telefonicaid/Rush)), a Docker file: [OrionRush on github](https://github.com/Geonovum/sospilot/tree/master/src/fiware/docker/orionrush)

To achieve best performance for the Orion context broker, we have to observe the tuning advices given in the [documentation on Readthedocs](http://fiware-orion.readthedocs.io/en/1.4.1/admin/perf_tuning/index.html)

## MongoDB Component

### Component Functions

[MongoDB](https://www.mongodb.com/) is a NOSQL document database, natively scalable by multiplying the nodes/containers running the software and replicating the data upon the nodes, this is called a replica set.

MongoDB is used for data persistence by several components of the platform: Orion context broker, IDAS IOT agents, Comet STH, Perseo.

### Technical Considerations

Each MongoDB container will use a data directory on the host system (outside the container) and mounted to a directory visible from inside the container.

It is recommended to use MongoDB version 3.2 or 3.4 and even 3.6 (the latest stable as of today).

From version 3.2 MongoDB uses a new optimized storage engine [WiredTiger](https://docs.mongodb.com/manual/core/wiredtiger/) (company owned now by MongoDB: [WiredTiger](http://www.wiredtiger.com/) open source storage engine)

The standard MongoDB incoming port is 27017/TCP.

Container RAM memory sizing: 4 GB

Storage sizing:

* Docker image size is 361 MB
* Data have to be stored on a specific Docker volume on the container host, each container running MongoDB should have its own volume on the host node (or reserved in the SAN).
* The size available for data storage, should be at least 300 GB for each storage volume

### Selected Product(s)

MongoDB is the selected NoSQL database for several Fiware components:

Orion Context Broker, IOT agents, Comet STH, Perseo

### Selection Rationale

As said, MongoDB NoSQL database engine chosen for several components proposed in Fiware.

MongoDB is a very popular scalable open source document oriented NoSQL database, that is horizontally scalable.

### Architecture Risks

MongoDB will be deployed as cluster in the global platform Swarm, to achieve high availability several replicas will be deployed (each replica is a container), see MongoDB documentation

Even built from origin to scale, and fault tolerant, MongoDB is not optimized for analytics, that uses cross data queries, by using a solution based on PostgreSQL, like [Stampede](https://www.torodb.com/stampede/) for Mongo slaves, that could be mapped to the global PostgreSQL/Citus cluster (using [cstore\_fdw](https://citusdata.github.io/cstore_fdw/)), high performance can be achieved on data reading and writing, while keeping long term history, then this is a solution to consider.

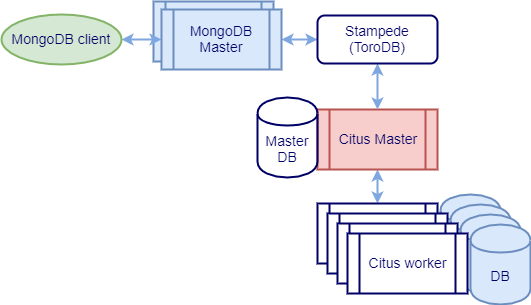


Figure 15: MongoDB with Stampede to Citus

## Citus/PostgreSQL cluster Component

### Component Functions

Citus provide a way to make a cluster of PostgreSQL database servers, it will serve database(s) to any component of the platform that need it.

### Technical Considerations

To reduce the number of different database servers in use in the platform, it has been decided to map components using an SQL database, to PostgreSQL, some component use MySQL as the default database, but container images will be changed to use PostgreSQL.

To achieve High Availability and Scalability of the database servers we’ll use Citus, to deploy a cluster of PostgeSQL databases.

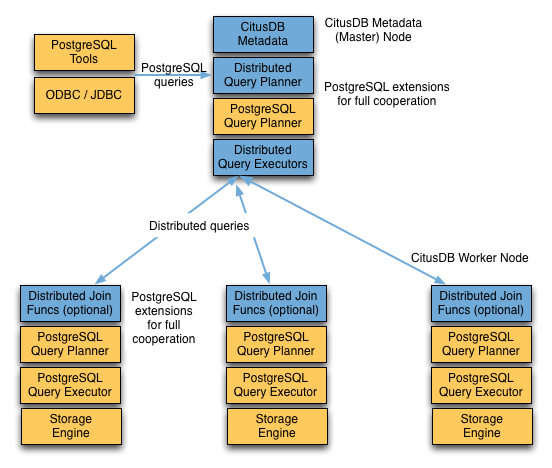


Figure 16: Citus high level architecture

Citus consist in a manager, a master that coordinate the cluster of Citus/PostgreSQL workers.

PostgreSQL standard incoming port is 5432/TCP, as well as Citus Master node which receives queries from clients.

Citus shall be installed through a Docker stack file: [how-to on Github](https://github.com/citusdata/docker)

### Selected Product(s)

Citus cluster for PostgreSQL is the selected product to provide Relational SQL database services to any application that needs it.

As some components are proposed configured to use MySQL database engine, the container images need to be reworked, to use PostgreSQL.

### Selection Rationale

PostgreSQL is the open source database that is the closest to Oracle in term of performance in traditional transaction based SQL applications, this at no cost.

But PostreSQL have also introduced NoSQL capabilities, that allows them to challenge tenors like MongoDB in the domain.

It is easy to add spatial capabilities to Postgresql, with PostGIS, well known and very popular spatial extension in the world of GIS.

[Several clustering solutions exist for PostgreSQL](https://wiki.postgresql.org/wiki/Clustering), such as [Postgre-XL](https://www.postgres-xl.org/) (a Postgres-XC fork/enhancement), among those solutions Citus has been chosen, because of its simplicity, stability and capabilities compared to the other solutions.

Citus is a PostgreSQL extension and not a fork, in sync with all the latest releases.

### Architecture Risks

As the cluster which provides PostgreSQL services for all the platform, Citus is a critical component.

To achieve best reliability and availability, the cluster should be deployed over the 3 MNCA data centers.

To achieve complete high availability the master needs to be replicated, the recommendation from CitusData is to use [PostgreSQL integrated Streaming replication](https://www.postgresql.org/docs/9.6/static/warm-standby.html#STREAMING-REPLICATION).

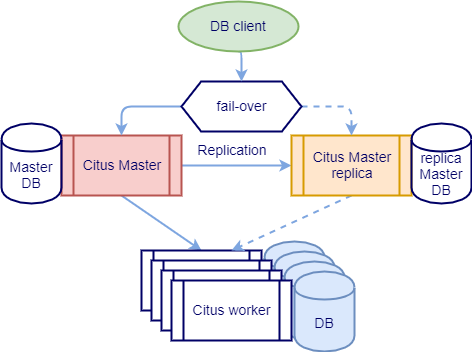


Figure 17: Citus/PostgreSQL cluster with HA master

## Galera/MySQL cluster Component

### Component Functions

Galera component is a cluster of MySQL/MariaDB databases.

### Technical Considerations

The cluster listen on standard MySQL port: 3306/TCP

### Selected Product(s)

This component shall be used only if some of components can’t be changed to use PostgreSQL

### Selection Rationale

Galera is a multi-master clustering solution for MySQL, as well for MariaDB

If a component is hard to change for using PostgreSQL instead of MySQL, the MySQL shall be replaced by a Galera cluster of MariaDB (full open soure), to prevent depending on MySQL, now owned by Oracle.

### Architecture Risks

The Galera cluster is a high available and scalable solution by conception.

## Cosmos Big Data Components, Cygnus & Comet

### Component Functions

Cosmos is the Big Data Analysis GE proposed in Fiware catalog,

[Cosmos](http://catalogue.fiware.org/enablers/bigdata-analysis-cosmos) is the Reference Implementation of the BigData Generic Enabler of FIWARE, a set of tools and developments helping in the task of enabling a Hadoop as a Service (HasS) deployment:

* A set of administration tools such as HDFS data copiers and much more, under [cosmos-admin](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-admin) folder.
* An OAuth2 tokens generator, under [cosmos-auth](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-auth) folder.
* A web portal for users and accounts management, running MapReduce jobs and doing I/O of big data, under [cosmos-gui](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-gui) folder.
* A custom authentication provider for Hive, under [cosmos-hive-auth-provider](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-hive-auth-provider).
* A REST API for running MapReduce jobs in a shared Hadoop cluster, under [cosmos-tidoop-api](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-tidoop-api).
* A specific OAuth2-base proxy for Http/REST operations, under [cosmos-proxy](https://github.com/telefonicaid/fiware-cosmos/blob/master/cosmos-proxy).

It’s role is also to store context data sent by the Orion context broker, to keep their history.

For that two components are used:

* Cygnus is a connector in charge of persisting certain sources of incoming data in some configured third-party storages, creating a historical storage of such data.
* The Short Time Historic (STH, aka. Comet) is a component of the FIWARE ecosystem in charge of managing (storing and retrieving) historical raw and aggregated time series information about the evolution in time of context data.
* Cosmos provide also a set of tools to access and use to an Hadoop environment.

As shown on the next Figure, Cygnus is used to store the historical data using output sink,

* Through the use of Mongo sink data are stored sample by sample.
* Through the use of STH sink data are store aggregated

Comet provides a RESTful API to query those historical data, either from the Mongo database filled by Cygnus Mongo sink, or from the Mongo STH.

The provided API is very close to NGSI, but with additional parameters allowing to query rough history or aggregated history, over a period (begin and end datetime)

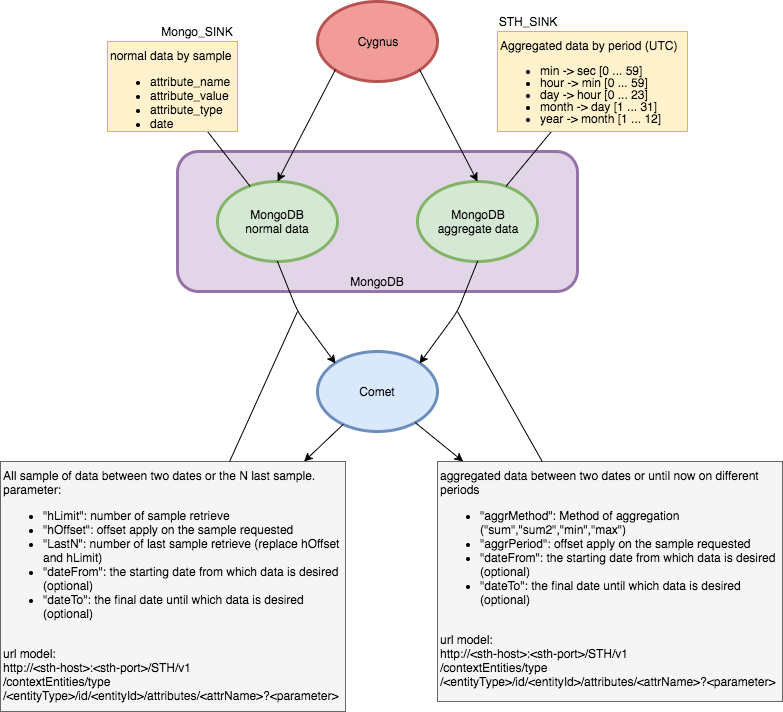


Figure 18 : Data instruction for Comet with STH-Sink and Mongo-Sink

The following figure shows the place of Cygnus in the Fiware ecosystem

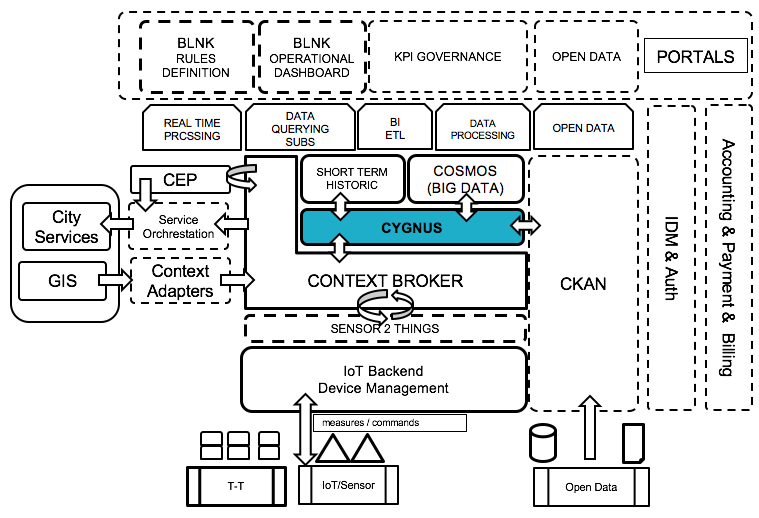


Figure 19: the place of Cygnus in Fiware architecture

### Technical Considerations

Cygnus is based on Apache Flume

It receives data form the Orion context broker, through the mean of subscriptions.

The subscription must be provided to Orion context broker, so that each time an update occurs on the context data concerned with subscription, Orion context broker propagate the change to the subscriber (in the case Cygnus, as the receiver of the NGSI context data.

When Cygnus receive data from Orion, it sends it to a database such as MongoDB or PostgreSQL, through a channel and a dedicated sink.

The following sinks are available in Cygnus to persist NGSI context data:

* [HDFS](http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html), the [Hadoop](http://hadoop.apache.org/) distributed file system.
* [MySQL](https://www.mysql.com/), the well-know relational database manager.
* [CKAN](http://ckan.org/), an Open Data platform.
* [MongoDB](https://www.mongodb.org/), the NoSQL document-oriented database.
* [STH Comet](https://github.com/telefonicaid/IoT-STH), a Short-Term Historic database built on top of MongoDB.
* [Kafka](http://kafka.apache.org/), the publish-subscribe messaging broker.
* [DynamoDB](https://aws.amazon.com/dynamodb/), a cloud-based NoSQL database by [Amazon Web Services](https://aws.amazon.com/).
* [PostgreSQL](http://www.postgresql.org/), the well-know relational database manager.
* [Carto](https://carto.com/), the database specialized in geolocated data.

Standard Cygnus incoming port: 5050/TCP

Comet provides API to retrieve data from the MongoDB where data are stored for STH (Short Time History ~ 1 year)

Standard Comet API incoming port: 8666/TCP

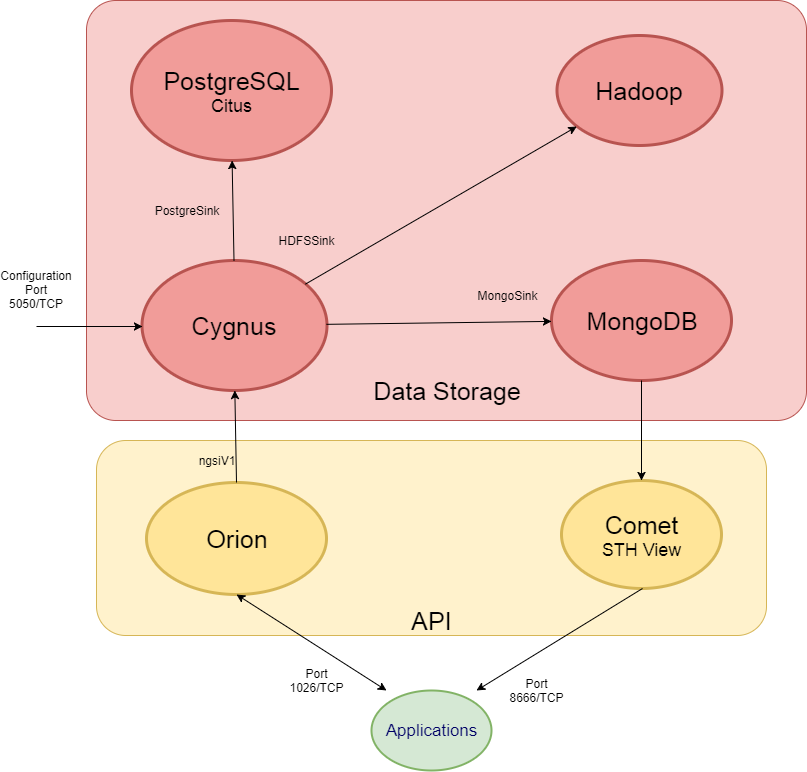


Figure 20 : Cygnus & Comet interactions

### Selected Product(s)

In a first step the following Cygnus sinks should be use:

* STHsink, to store already aggregated data in the MongoDB database cluster as specific collection.
* Mongosink to store raw context data history in the MongoDB database cluster as specific collection.

In a second step, either it’s considered necessary to setup an Hadoop platform and use the HDFS sink of Cygnus, but before it could be valuable to test the performance of using stampede behind MongoDB, but with data landing on the Citus cluster of PostgreSQL databases.

By using proper sharding keys, to partition the data history, for which Citus is optimized for, it is possible to wait before diving in the Hadoop ecosystem (HDFS, MapReduce, Hive…)

### Selection Rationale

Cygnus and Comet are the reference implementation of Big Data Component in Fiware catalog.

Cygnus can easily send data to another storage like Hadoop if needed.

The currently provided sinks cover a large palette of choice for data history storage.

But when it comes to the use of Hadoop, the only sink available is HDFS, where context history is stored in files (one file per history event), and the hierarchy is given by the file path (<service>/<sub-service>/<entity ID>), this means that the storage is not indexed on data, and then the read performance should be very low !

If a Cygnus NGSI sink to a HDFS based NoSQL database like **HBase** appears (this was envisaged earlier in the project), it could be better, and

The proposal to wait before setting up an Hadoop, doesn’t mean it’s impossible to analyze data, and provide pertinent visualization on it, the use of the **Knowage** analytic component provides the panoply of tools to access any Big Data source such as Hadoop and many NoSQL databases like MongoDB, in addition to traditional relational database like PostgreSQL, MySQL.

In addition, using recent modern analytics engine that works in memory such as [Apache Spark](https://spark.apache.org/), outperforms from 10 to 100 times compared to a Hadoop MapReduce solution depending on the type of processing.

Moreover, Spark solution, can source the data to analyze from traditional or NoSQL databases, then if the target databases services are optimized for big data analytics such as Citus is, it’s double win.

### Architecture Risks

As the component to manage data history storage, Cygnus and Comet need to be high available.

As stateless components, it is easy to run several instances of the service by running several containers, it provides HA, but also scalability, Docker through ingress network, perform load balancing between containers of a service.

## Knowage Analytics Component

### Component Functions

Knowage is a professional open source suite for modern business analytics over traditional sources and big data systems.

Knowage is the new generation of open source analytical solution, as a natural evolution of the well-known SpagoBI.

The features of the component are:

* BD (big data), to analyse data stored on big data clusters or NoSQL databases
* SI (smart intelligence), the usual business intelligence on structured data, but more oriented to self-service capabilities and agile prototyping
* ER (enterprise reporting), to produce and distribute static reports
* LI (location intelligence), to relate business data with spatial or geographical information
* PM (performance management), to manage KPIs and organize scorecards
* PA (predictive analysis), for more advanced analyses
* EI (embedded intelligence), to link Knowage with external solutions provided by the customer or third parties.

Knowage can connect to CKAN and show the catalog of data sets in it, allows to search in it, to be able to use it (only CSV & XLS data source are supported for the moment, other formats will be added later).

Knowage can also define an NGSI dataset (query data from the Orion Context Broker).

Knowage can use common big data NoSQL databases such as MongoDB and Hadoop based environments as data sources,

Moreover, it can be used in conjunction with Apache Spark to analyze data set from 10 to 100 times faster than with Hadoop Mapreduce.

Knowage SDK contains a Javascript API that helps users to embed parts of Knowage suite inside a web page or to retrieve information about datasets and documents.

### Technical Considerations

The component will be configured to use the Keyrock IDM component for authentication.

Knowage is based on Tomcat 7 and MySQL, the container image will be changed to use PostgreSQL, by changing the Hibernate ORM configuration used in the software.

Incoming port default to 8080/TCP conflict with other components using java application servers, it can be easily changed with Docker

### Selected Product(s)

This is the reference component for data visualization and BI in Fiware catalog

It replaces the previous GE: SpagoBI, from which Knowage is an evolution.

### Selection Rationale

Knowage takes part in the FIWARE community, being the reference implementation of the Data Visualization GE

### Architecture Risks

If the component is used for help to operational decision, its high availability becomes mandatory.

The component is stateless, by adding instances of containers, and relying on a Citus database cluster for data persistence, high availability and scalability are easy to achieve

## CKAN Open Data Component

### Component Functions

Open Data management system

### Technical Considerations

PostgreSQL database

Server written in python

Management Web interface

RESTful API

Security (Authentication, Authorization provided by IDM Components)

### Selected Product(s)

### Selection Rationale

### Architecture Risks

## IDAS IOT Components

### Components Functions

IDAS is composed of several components dedicated to IOT, these components are responsible to ingest the data coming from the field, transform it into NGSI entities data, and to update the context broker.

On Fiware Catalog: [IDAS](https://catalogue.fiware.org/enablers/backend-device-management-idas)

Documentation on the Fiware IOT stack : [by Telefonica](https://fiware-iot-stack.readthedocs.io/en/latest/)

Available IOT Agents:

* Ultra Light IOT agent ([UL2.0 IOT Agent](https://github.com/telefonicaid/iotagent-ul)), is a bridge that can be used to communicate with devices. The UL2.0 protocol is optimized to communicate with limited resources. This IOT Agent can bind different protocols to UL2.0 like HTTP or MQTT. example of request: “t|25” to change an attribute bind on t like “temperature”.
* [JSON IOT Agent](https://github.com/telefonicaid/iotagent-json), can also bind different protocols like HTTP or MQTT, it allows to convert incoming device data coming as JSON to be converted to NGSI context data.
* SigFox IOT Agent: [SIGFOX IOT Agent](https://github.com/telefonicaid/sigfox-iotagent) on github
* LWM2M IOT Agent: [OMA LWM2M IOT Agent](https://github.com/telefonicaid/lightweightm2m-iotagent)

All IOT agents are based on the same architecture, provided by a core common library:

[IOT Agent library used to develop other agents](https://github.com/telefonicaid/iotagent-node-lib)

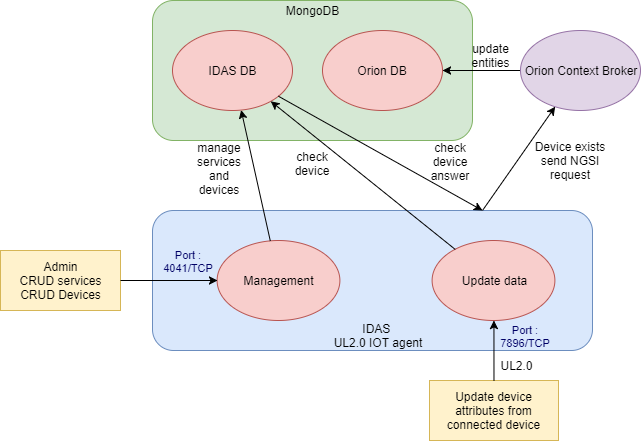


Figure 21 : IDAS UltraLight IOT Agent interactions

All IOT agents works the same way as UL2.0 IOT Agent., relying on the core library provided.

Each IOT agent provides device and service management features through a RESTful API.

This management interface allows to configure the conversion between the device data and the NGSI entity to update in the context broker, for UL2.0 and JSON data sources, also are available LWM2M IOT agent, Sigfox IOT agent.

In the case of MNCA, the existing data sources doesn’t use any of those protocols, data come either from xml, csv or even text files, retrieved or pushed through FTP.

A few data source are updated through periodic http requests, that extract & convert data before feeding the existing data warehouse.

Those data are currently inserted in the data warehouse, using an ETL step and/or the SCOWS api.

Current data sources are:

* Environment data Nice & Cagnes/Mer 38 Stations
* Air quality monitoring 8 Stations
* “Météo France” weather forecast 17 forecast Stations
* Weather Observations from Infoclimat 4 Stations
* Weather Observations WeatherCompany 12 Stations
* Hydraulic Station Data - SAC Magnan 4 Stations, SAC Paillon 13 Stations, East Mediterranean Flood Forecasting Service 11 Stations
* Statement of Voluntary Contribution Points 211 Stations
* Measure of urban traffic 327 Stations
* Tourism 9 data sets
* Parking 13 parkings
* Transport - Tramway 28 tramways T1 (futures 3 times more with T2, T3)
* Forecast water metering through LoRa 60 000 devices

Those data sources are under study in a parallel project, the aim is to materialize NGSI representation of the different data sources, by trying to map on existing Fiware data models, and propose evolution to those models, when needed, through the process proposed by Fiware.

To be able to ingest those existing data sources in the new platform, it is necessary to either develop some specific IOT agents, using the core library, but after reflection it has been thought better to use the JSON IOT agent behind an MQTT broker, for the communication with the context broker.

The data sources are prepared from files, or http request, through specific programs, where data is converted in JSON, and published to the MQTT broker topic, on which the JSON IOT agent have subscribed.

The following schema shows the envisaged architecture for the data acquisition pipeline.

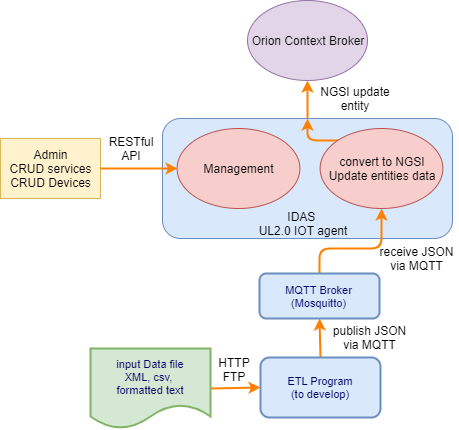


Figure 22: Existing data sources import overview

This group of modules is the object of a specific project started in parallel.

Depending on the targeted data source mode of acquisition (FTP get, NFS file access, http get), several modules should be deployed as containers, it could be either some permanent module waiting for a file to appear in a particular directory, or a scheduled script that query the data through http or ftp, before processing it.

For the moment the following components are still missing in the Fiware catalog:

* The IOT Agent Manager, is under porting to Node.js (scheduled for R5 release), should allow to manage IOT agents, through a RESTful api.
* IoT Edge Manager (forthcoming for R5 release) should allow to manage some sensors network, and communicate directly to the context broker using NGSI api.

As seen through the document, the Fiware platform is a collection of RESTful API, characteristic of a micro service architecture.

So, the platform is a big lego, that nevertheless still needs to be assembled and exposed to users through ergonomic human-machine interface, such as a web application to manage easily: services, devices, credentials, IOT Agents, context entity, and several other functions to give an easy global way to manage IOT concerns like provisioning, security

Telefonica have tried to provide such an application through an [IOT Orchestrator with a portal UI](https://github.com/telefonicaid/orchestrator)

But the portal application is not open source, and the project is not really maintained, recent changes removed the possibility to get the portal Docker image.

To summarize this global IOT management application should be designed and developed, taking example on the Telefonica Portal.

A specific paragraph

### Technical Considerations

IOT Agents provide two main interfaces:

A management interface through RESTful API

An interface to receive data from devices/sensors or gateway data, which support either HTTP or MQTT transport.

So, each IOT agent require 2 open incoming ports.

For MQTT standard incoming ports of the broker are:

* 1883/TCP for unsecure communication channel,
* 8883/TCP for secured SSL/TLS channel

### Selected Product(s)

In a first step, only the JSON IOT agent component will be used to ingest existing data sources into the Fiware platform.

The MQTT broker can be Mosquitto or any other message broker supporting MQTT, such as RabbitMQ or ActiveMQ, in a fisrt step Mosquitto has been chosen.

The components proposed in the IDAS part of the Fiware catalog are now developed with Node.js, this is the case for all the IOT agents.

The database used by the IDAS components is MongoDB.

The first stage “ETL” modules will be developed using [Node-RED](https://nodered.org/)

### Selection Rationale

The tools proposed in the Fiware catalog has been chosen.

Many IOT agents are already available for different transport and protocols, some are under development like an OPC UA IOT agent, which open the way to connect directly SCADA or even industrial PLCs.

The use of a core IOT agent library written in Node.js as the base for all others agent is a good architectural choice.

As well as the choice of MongoDB for the data persistence.

To complete the needs for data ingestion, Node-RED shall be used to provide the first level of conversion of data sources in JSON, before posting it to the MQTT broker, to be consumed by the JSON IOT agent.

### Architecture Risks

IOT Agents must be deployed targeting high availability and scalability in production, this can be easily achieved by adding some instance using a Docker service stack file, since IOT agents are stateless components.

The MongoDB database they rely on, is a cluster providing high availability and scalability as well.

The security between the devices and the IOT agent communicating through HTTP rely on the use of an API key, that is checked by the IOT agent to accept the incoming message, the security can be enhanced through the use of HTTPS.

When the transport is done through MQTT, the secure mode using SSL/TLS need to be forced to use, then any client needs to provide valid credentials to be authenticated and authorized, to use the MQTT broker publish/subscribe services.

## CEP Component

### Component Functions

CEP stand for Complex Event Processing, it has not been invented for IOT, but originally to automate business decision.

### Technical Considerations

### Selected Product(s)

Telefonica Perseo or IBM Proton

At a first glance Perseo is simpler to use and manage

### Selection Rationale

In Fiware catalog is proposed an implementation from IBM of the CEP module: [PROTON](https://catalogue.fiware.org/enablers/complex-event-processing-cep-proactive-technology-online)

It is a scalable integrated platform to support the development, deployment, and maintenance of event-driven applications.

Telefonica propose also an implementation of the CEP GE: [Perseo](http://fiware-iot-stack.readthedocs.io/en/latest/cep/), as the CEP GE of their IOT stack

As well as Orange which propose [Cepheus](https://catalogue.fiware.org/enablers/iot-data-edge-consolidation-ge-cepheus)

It’s composed of a light NGSI broker and a CEP which features a dedicated REST management API to configure real time analysis on NGSI events.

Either Perseo and Proton has been tested by Faycal Boufares, it appears that Proton is richer, but more complex to apprehend than Perseo.

Perseo use a MongoDB database, Proton uses files to store events configurations

.

### Architecture Risks

Perseo intercepts all request incoming to Orion Broker, in case of high traffic it can become a bottleneck that delay the processing of important rules.

## Application Mashup Components

### Component Functions

Those components allow to design and publish dashboards and visualization mashups about entities data from the context broker and the short-term history, but without the need of programming skills.

The components provide simple interface to assemble in a page, configurable widgets to visualize entities data.

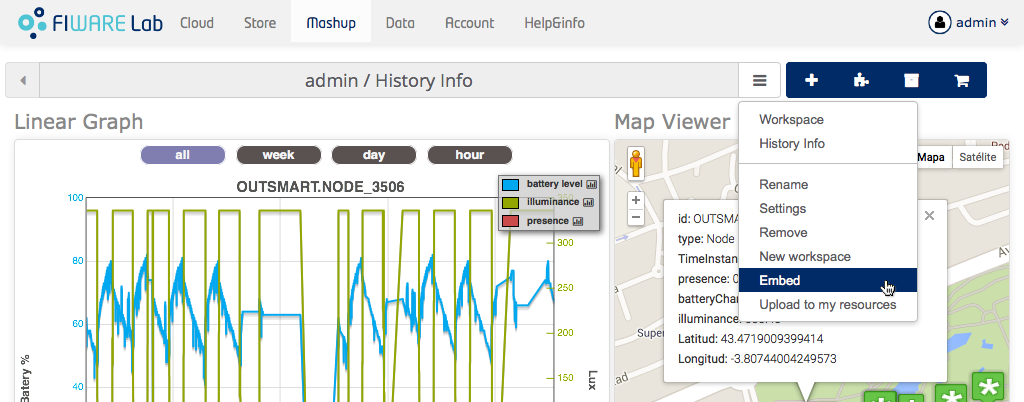


Figure 23: Wirecloud Mashup example

### Technical Considerations

Wirecloud composed of a back-end server and a javascript web application,

the Wirecloud back-end server is based on Django Python application server, it uses a PostgreSQL database (or MySQL, SQLite3), to persist mashups and dashboards, in our case it will be the Citus cluster of PostgreSQL databases.

Wirecloud can be easily deployed as Docker containers: [instructions on github](https://github.com/Wirecloud/docker-wirecloud/) image on [DockerHub](https://hub.docker.com/r/fiware/wirecloud/)

Freeboard is a simple javascript library that can be deployed in any web server, or even embedded in a larger application, as it does not provide any security, it’s the best solution.

Freeboard persists dashboard in JSON files on the web server where it’s installed

Freeboard can be installed through Docker from an NGINX base image: [on GitHub](https://github.com/linaro-technologies/freeboard-docker)

But with this image, the NGSI connector is not installed, it can be found [on Github](https://github.com/telefonicaid/fiware-connectors/tree/master/FreeBoard-Orion-Plugin):

(Orion\_DataSource is a javascript plugin to feed a Freeboard dashboard with NSGI Context Brokers such as Orion).

### Selected Product(s)

The 2 selected components for application mashups and dashboards are: Wirecloud and Freeboard.

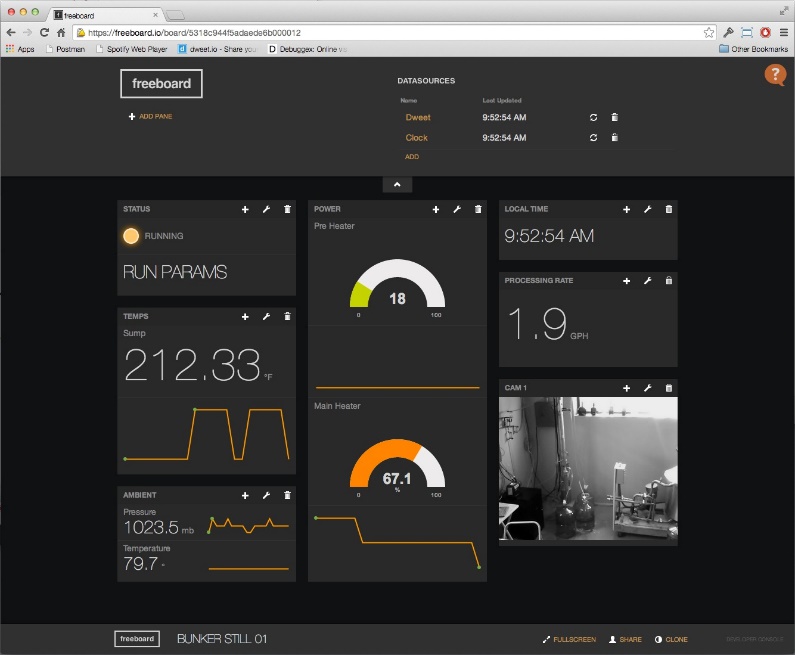


Figure 24: Freeboard Dashboard example

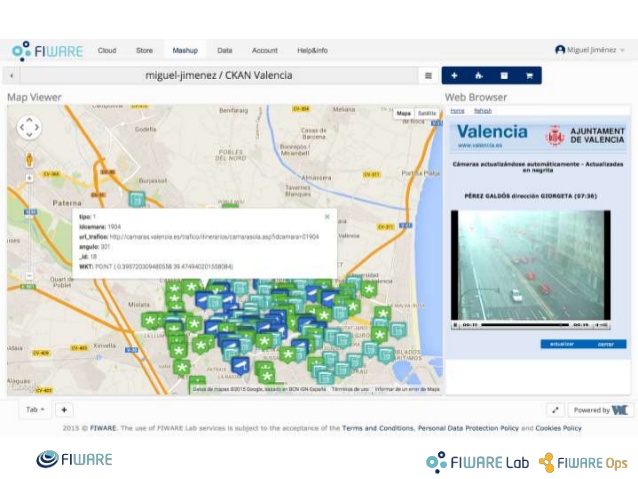


Figure 25: Wirecloud Mashup example

### Selection Rationale

Wirecloud and Freeboard are the components proposed in Fiware catalog for mashup applications.

Both components provide an NGSI data source interface, to retrieve context information from Orion Context Broker.

In addition, Wirecloud provides an interface to query the STH component to get data history.

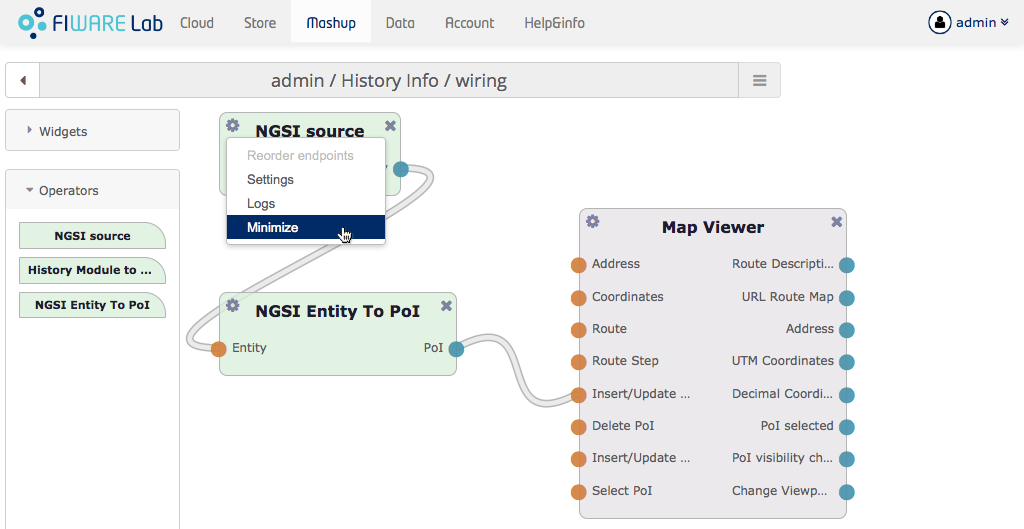


Figure 26: Wirecloud graphic configuration interface

### Architecture Risks

If those components are used to deliver some important information through dashboards to help for decision, they need to be high available.

For Wirecloud, as well for Freeboard, should run as Docker services to provide HA, several instances of containers can be started for scalability.

The database used by Wirecloud is the Citus cluster, then provide HA for the persistence layer.

## Business Ecosystem GE Components

### Components Functions

The Business Ecosystem GE is composed of several modules that cover the following features:

**TM Forum Apis:**

* Catalog management,
* Ordering management,
* Inventory management,
* Usage management,
* billing,
* customer,
* party APIs.

**Business Ecosystem Charging Backend:**

* Process the different pricing models, the accounting information, and the revenue sharing reports.
* Calculate amounts to be charged, charge customers, and pay sellers, using this above information.

**Business Ecosystem Revenue Sharing Service:**

Distributes revenues originated by the usage of a given service among the involved stakeholders.

It distributes part of the revenue generated by a service between the Business API Ecosystem instance provider and the Service Provider(s) responsible for the service, a service here refers to both final applications and backend application services.

**Business Ecosystem Logic Proxy:**

Is the endpoint for accessing the Business API Ecosystem:

* it orchestrates the APIs validating user requests, including authentication, authorization, and the content of the request from a business logic point of view.
* it serves a web portal that can be used to interact with the system.

This is the Market Place component of the smart city platform, that allows to monetize applications, APIs and also data that are exposed through the smart city platform open data component such as CKAN,

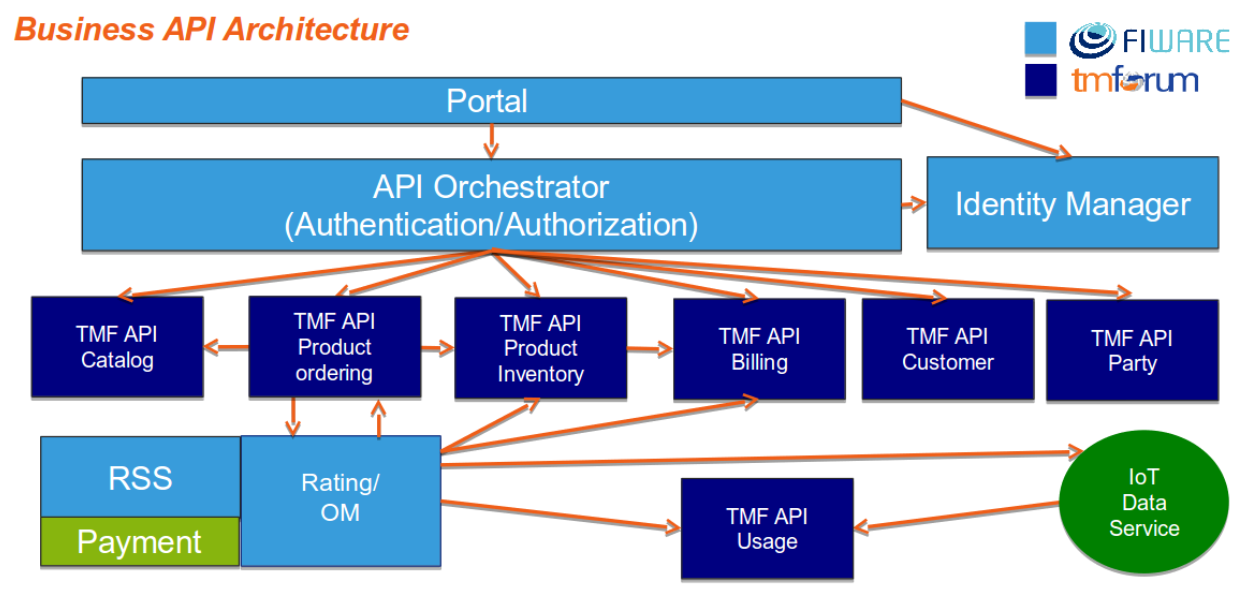


Figure 27: Fiware Business Ecosystem Architecture

Applications to monetize can include: dashboards made through WireCloud or Freeboard, or any application developed using Eclipse Che PAAS.

### Technical Considerations

The Business ecosystem GE, is composed of many components, that work together and interact with other components depending on the kind of business model needed.

The components of the GE are:

**TM Forum APIs and Revenue Sharing Service (RSS) requirements**

* Java 8
* Glassfish 4.1
* MySQL 5.5 => to replace with PostgreSQL/Citus (or in extreme case use Galera Cluster and MariaDB if hard to change from MySQL to PostgreSQL)

**Charging Backend requirements**

* Python 2.7
* MongoDB
* wkhtmltopdf

**Logic Proxy requirements**

* NodeJS 4.5.0+ (Including NPM)

Memory use depends on the number of concurrent users as well as the free memory available and the hard disk. The Business API Ecosystem requires a minimum of 1024 MB of available RAM memory, but 2048 MB of free memory are recommended.

Incoming Ports:

Disk usage: Business API Ecosystem requires at least 15 GB of hard disk space.

Database sizing: to be evaluated through a use case to estimate the DB size

### Selected Product(s)

The other components proposed for business management have been deprecated, in favor of TM Forum API and tools, which are now the proposed GE in the Fiware catalog.

### Selection Rationale

This is the reference component in Fiware Catalog, since the collaboration between Fiware and TM Forum have started, the Business Ecosystem GE, has been completely reworked, and now rely on TM Forum APIs, and new components.

The use of TM Forum APIs is a warranty of being able to conduct easily business relations with different stakeholders.

### Architecture Risks

Is to be consider as a critical component, because it mediates the usage of applications and their APIs.

The services need to be high available

## Development Components

### Component Functions

Development IDE is Eclipse Che, it allows to develop natively Docker based applications, it should play the role of the PAAS of the MNCA Smart city platform.

The Eclipse Che IDE provides a web based interface to developers, so that they just need to use a browser connected to the Che server

SCM and project management should be managed with the already installed [MNCA GitLab](https://gitlab.nicecotedazur.org/)

### Technical Considerations

Eclipse Che is a java8 (using openjdk) program running under Tomcat application server.

The component can be very easily install with Docker

The Eclipse Che server listen on 8080/TCP port (Tomcat standard)

It also provides a RESTful API endpoint on port 8000/TCP.

Eclipse Che can be used in multi-tenant mode, where users have to connect with credentials to access the IDE.

In standard installation it uses Keycloack as IDM provider, and a PostgreSQL database for metadata.

As Keycloak can replace Keystone as an authentication service (OAuth provider), in OpenStack deployment, and Eclipse Che can also use OAuth from GitLab or WSO2, if needed Keyrock can be also used as the OAuth provider giving SSO to developers using the Eclipse Che server.

Keycloak comes also with a built-in LDAP/AD plugin, which supports password validation via LDAP/AD protocols and different user metadata synchronization modes, so that internal users already registered on MNCA ADAM, can be identified in Eclipse Che.

### Selected Product(s)

Eclipse Che as the PAAS IDE environment

GitLab for SCM and project management

### Selection Rationale

GitLab is already in use in MNCA to manage development project and source code, it’s a ready to use git (SCM) with project management features, it allows to manage groups & users, project & issues (working tasks and bugs).

Eclipse Che is not part of the Fiware Catalog, but it’s one of the most advanced development environment to develop Docker based applications.

### Architecture Risks

If used in multi-tenant environment, Eclipse Che should be deployed high available, through a Docker service stack.

## Monitoring Components

### Component Functions

The monitoring components are very useful and mandatory to check the health of the platform.

The hardware is monitored in term of CPU, Disk, memory and network.

As well containers are monitored for CPU, RAM, Disk and network usage.

The different services provided by the platform need to be monitored too:

* Database services are monitored for health, CPU, memory, number of active connections, I/O activity and any additional useful information (like transactions performance)
* Other services are also checked for health, and monitored about CPU, memory, network and I/O activity.

Apart from monitoring the services, the management of their logging files is also needed for several reasons:

To be able to detect when errors occur and send notifications about it.

To be able to view and analyze all the log files from one single point.

### Technical Considerations

Portainer is made for managing Docker containers and support swarm, it installs as a standalone container.

The container image size is 11 MB

Memory usage: 10 MB minimum

It needs to be mapped to a docker volume outside the container, to keep the configuration data.

Centreon is already in use at MNCA to monitor the entire IT infrastructure, if needed a new instance of Centreon can be installed through Docker container: [Centreon Docker installation](https://github.com/julienmathis/centreon-docker)

For the management of log files Docker should be configured to use a gelf logging driver:

### Selected Product(s)

Real Time monitoring is done with Portainer.

Monitoring with history and notification will be provided through Centreon.

For the management of the log files produced by components, [Docker supports many different logging drivers](https://docs.docker.com/engine/admin/logging/overview/#supported-logging-drivers): default is json-file, but Journald or syslog can be used.

the most interesting option should be to use [gelf logging driver](https://docs.docker.com/engine/admin/logging/gelf/), which writes log messages to a Graylog Extended Log Format (GELF) endpoint such as Graylog or Logstash, using Logstash the logs can be pushed into an ElasticSearch instance for analysis.

### Selection Rationale

Portainer is very simple to install and use

Centreon is already used for monitoring the IT infrastructure and the services.

### Architecture Risks

# Section 4 System Construction Environment

*The* ***System Construction Environment*** *section details the various environments necessary to enable system construction and testing.*

*For each environment necessary for system construction (****Development, QA*** *and* ***Acceptance****), provide detailed specifications for the* ***Environment*** *and* ***Supporting Infrastructure*** *that will be used (including hardware*, network *and operating system requirements, all necessary installed packages and tools, and needed directory structures**that will be utilized to store all construction components).*

## Preliminary test Environment

A preliminary test environment has been setup, based on 4 virtual machines

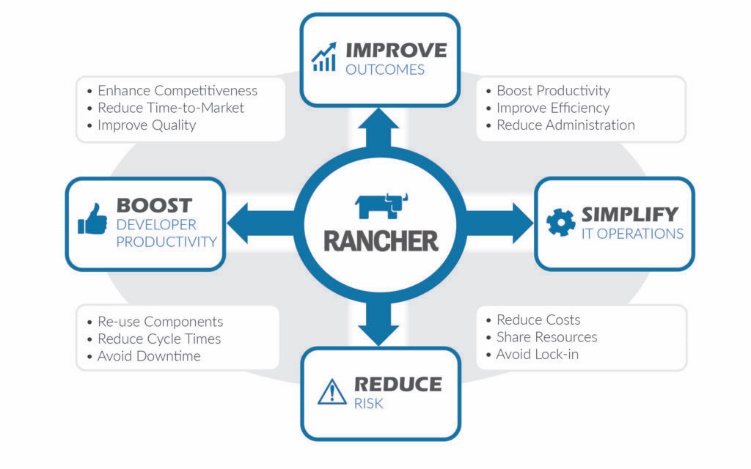
The available components are:

* Authentication portal Horizon: [iot.testnca.org:8000](http://iot.testnca.org:8000)
* Authentication API Keystone: [iot.testnca.org:5000](http://iot.testnca.org:5000)
* Sécurisation de l’API ORION via PEP-Proxy: [iot.testnca.org:8081](http://iot.testnca.org:8081)
* API Orion Context Broker: [iot.testnca.org:1026](http://iot.testnca.org:1026)
* API IDAS http UltraLight: [iot.testnca.org:4041](http://iot.testnca.org:4041)*(Service/Device Creation)* [iot.testnca.org:7896](http://iot.testnca.org:7896)*(http UL2.0 for data update)*
* MongoDB (3 containers): [iot.testnca.org:27017](http://iot.testnca.org:27017)**, iot.testnca.org:27018, iot.testnca.org:27019**
* CKAN Opendata Portal: [iot.testnca.org:8002](http://iot.testnca.org:8002)
* Wiki Documentaire Dokuwiki: [iot.testnca.org:8003](http://iot.testnca.org:8003)
* A portainer is also available:[iot.testnca.org:9000](http://iot.testnca.org:9000)
* A Docker registry has been installed: **registry.nicecotedazur.org:5000** (portus ?)
* An Eclipse Che has been installed in test: [iot.testnca.org:8080](http://iot.testnca.org:8080) (url to provide che.nicecotedazur.org? ;O)

## Global system vision

The targeted global system should offer several environments to cover the whole life cycle of software from development, to QA, and then to production.

It could be valuable to introduce meta-orchestration tools like [Rancher](http://rancher.com/), to be able to manage easily the different environments.



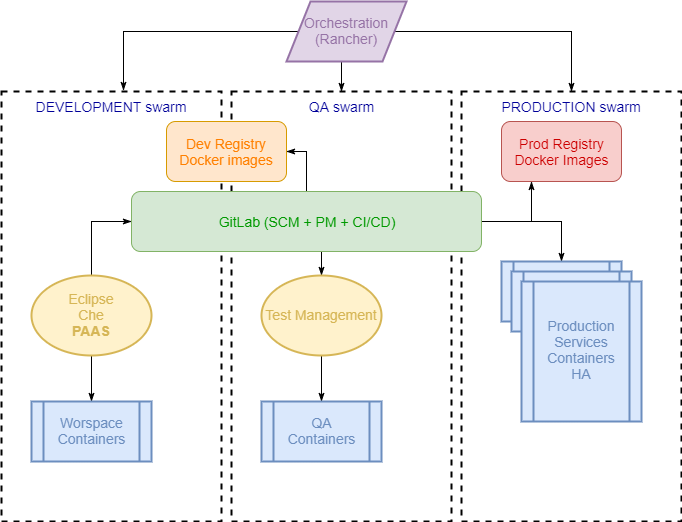


Figure 28: global environment logical big picture

## Development Environment

### Developer Workstation Configuration

By using natively Docker to develop applications with eclipse Che, developers have only to use a web browser to perform their work, they should have access to the associated GitLab to push the code.

### Supporting Development Infrastructure Configuration

Development infrastructure should be composed of at least 3 nodes, one in each data center.

Adding nodes in case of need to scale-up the capacity is very easy with swarm.

The main supporting component for development is Eclipse Che

## QA Environment

### QA environment Configuration

The QA (Quality Assurance) environment is to be set for testing purpose.

### Supporting QA Infrastructure Configuration

## Production Environment

### Production environment Configuration

### Supporting Production Infrastructure Configuration